ALGORITHMS AND PROTOCOLS FOR NEXT GENERATION WIFI NETWORKS

A Thesis
Presented to
The Academic Faculty

by

Chao-Fang Shih

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2017

ALGORITHMS AND PROTOCOLS FOR NEXT GENERATION WIFI NETWORKS

Approved by:

Professor Raghupathy Sivakumar,
Advisor
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Professor John R. Barry
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Professor Douglas M. Blough
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Professor Karthik Ramachandran
Scheller College of Business
Georgia Institute of Technology

Professor Faramarz Fekri
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Date Approved: November 29, 2016

To everyone I met through my journey.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Raghupathy Sivakumar, without whom this dissertation would not have been possible. With the high standard of research quality, broad vision in wireless communication, clear thoughts, effective communication, strong enthusiasm, and great patience, he has served not only as a mentor but also a role model of a researcher for me.

Second, I would like to thank Prof. Douglas M. Blough, Faramarz Fekri, John R. Barry, and Karthik Ramachandran for serving on my proposal and dissertation committee and devoting time to provide constructive feedback.

My gratitude extends to the members of the GNAN research group. I thank Sriram Lakshmanan, Sandeep Kakumanu, Cheng-Lin Tsao, Shruti Sanadhya, Bhuvana Krishnaswamy, Uma Parthavi Moravapalle, Yubing Jian, and Mohit Agarwal for their valuable support, feedback, and assistance in my dissertation.

Last but not the least, I would like to thank my family and friends. Their encouragement, love, sacrifice and support have helped me go through the ups and downs of this journey. I owe all my success to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The internet traffic requirement has grown remarkably in the last couple of decades, and it is predicted to scale even further in the upcoming years. On the other hand, internet users are increasingly relying on WiFi for the last mile connectivity. WiFi has been used in a broad range of applications not only in enterprises but also in our daily lives. Thus, technologies that can fundamentally enhance WiFi are desirable. Research on next generation WiFi networks is focusing on a variety of goals including performance (i.e., system throughput and channel efficiency), controllability, security, power consumption, adaptability, etc. The goals of our research are improving performance to support the growth of requirements, and promoting controllability to support service differentiation and even performance prediction. Performance goals pertain to achieving better throughput and channel efficiency for a wide range of network conditions with the highest degree of adaptability. Controllability goals are akin to the goals of software-defined networks and focus on future-proofing systems, manageability, and service differentiation capabilities.

First, we propose algorithms that improve the performance in the physical (PHY) layer with only changes in Access Points (APs). Smart antenna technology plays an important role for performance enhancement in the PHY layer of WiFi. While most smart antenna techniques require changes in both APs and stations (STAs), beamforming is a mechanism that can be applied with only changes in APs. We propose *FastBeam*, a set of algorithms that can provide benefits of beamforming to legacy nodes by only adopting new APs. FastBeam focuses on heavy multipath fading indoor environment (such as enterprise offices or school classrooms), which is a typical environment for legacy WiFi infrastructure.

Second, we consider future-proofing networks with a central controller and enable

micro-level controllability to support service differentiation and performance prediction. We focus on how to *enable the controllability of WiFi networks without compromising their scalability* when a central controller is available. We introduce a media access control (MAC) protocol called *Rhythm*, which transfers the control of WiFi networks into centralized scheduling, with the properties of (i) low protocol overhead, (ii) work conservation in the presence of non-backlogged nodes, (iii) robustness to partial connectivity scenarios. Specifically, Rhythm furnishes all nodes in WiFi networks with a *target schedule* that has been determined by a central controller. The nodes in WiFi networks then operate in a *purely distributed fashion* to meet the target schedule. We refer to such a system behavior as "*scheduled WiFi*."

Finally, we consider the backward compatibility issues of scheduled WiFi. Though Rhythm brings micro-level controllability to WiFi, it can only operate in an ideal environment, where i) no other legacy nodes are present and ii) the network topology is known. To provide better backward compatibility for implementing *scheduled WiFi*, we propose a new MAC protocol, *LWT*. LWT not only achieves scheduled WiFi in a purely distributed fashion, but is also more friendly to legacy nodes, and does not require the network topology information.

LWT uses a novel mechanism, called *Switch*, to deal with hidden terminal problems. Switch provides a way to utilize overlay control channels better in WiFi networks. In addition to the details of how Switch is used in LWT, we further explored how Switch can be used in other situations, such as i) extending the range of carrier sense, ii) early collision termination, and iii) improving the efficiency of WiFi backoff mechanism.

CHAPTER I

INTRODUCTION

The last couple of decades have seen a remarkable growth in the number of Internet users. Over 75% of the population in the United States now use the Internet [7] for various applications. The ubiquitous availability of broadband access through both cable companies (cablecos) and telecommunication companies (telcos) has been a primary driver of the high rate of adoption. Users increasingly depend on the Internet for a wide variety of rich content and services. Network access speeds thus have to scale with the demands. Data rates offered through cablecos, for example, have raised from 1.5Mbps in the year 1998 to 305Mbps in the year 2013 [2], which is a 200x increase in the span of 15 years. However, there is still an increasing pressure to scale speeds even further. Recently, the offering of 1Gbps rates [3] and evolving technologies such as DOCSIS 3.1, which allows for 10Gbps rates ([5]), are evidence of this pressure of cablecos and telcos.

On the other hand, consumers at homes and enterprises are increasingly relying on WiFi connectivity for the last mile between the network backbone and the end-user devices. We refer to this connection as the $zero^{th}$ mile. Over 80% of US homes with broadband access rely on WiFi for in-home connectivity. Enterprises are not different. Gartner projects that by 2018, more than 50 percent of users will first go to a tablet or smartphone for all online activities. These mobile devices, in turn, rely on WiFi for network connectivity. Perhaps more surprisingly, Gartner also predicts that by the same year (2018), 40 percent of enterprises will specify WiFi as the default connection for non-mobile devices, such as desktops, desk phones, projectors, and conference rooms [6].

Overall, more than 50% of Internet traffic is expected to be from or to WiFi devices by 2019. Furthermore, next generation WiFi is expected to power the emerging field of

Internet of Things (IOT). All these trends indicate that WiFi becomes not only a dominant technology in the Internet landscape but also a fundamental basis for most of the Internet traffic. This phenomenon has exposed an interesting conundrum. Even though the wireline speeds are substantial with 1Gbps access network speeds deliverable today and 10Gbps achievable in the near future, what the user experiences is heavily limited by the data-rates on the $zero^{th}$ mile.

This situation has unsurprisingly led to intensive research into how WiFi can be made to operate better and deliver higher performance for a wide variety of applications. Below, we briefly introduce different research works according to the provided enhancement in WiFi:

- Improve the PHY layer throughput of WiFi: Through the last decade, WiFi protocols have mainly enhanced the performance of the PHY layer (802.11n [20], 802.11ac [21]). The usage of wider bandwidth, higher modulation rate, the introduction of smart antenna techniques (beamforming and Multiple-Input Multiple-Output (MIMO)), utilization of more antennas, and more powerful hardware give tremendous PHY layer throughput improvement from 54Mbps to 7Gbps.

- Improve the MAC layer throughput of WiFi: The MAC protocol used by most WiFi networks is a distributed MAC protocol, Distributed Coordination Function (DCF). A distributed MAC protocol has many benefits including low complexity, scalability, and robustness to dynamic traffic loads. However, its channel efficiency decreases as contention levels increase. Considering the current growth of the number of WiFi networks [8], algorithms that try to improve the MAC layer throughput under high contention levels have been proposed [75, 36].

- Increase the macro-level controllability of WiFi: To fulfill the diversity of service requirements, researchers are devoting considerable attention to *improving WiFi network controllability without compromising its essential scalability.* WiFi controllers

that centralize the management of APs within an enterprise network may be the simplest example of such efforts. Functionalities such as radio resource management, wireless policy management, and authentication services are moved to the central controller [18, 19]. However, while WiFi controllers allow for reconfigurability, they do so only for macro-level "control plane" parameters such as authentication settings and channel assignments, and they do not control micro-level "data plane" functions such as media access control.

- Increase the micro-level controllability of WiFi: Some protocols and algorithms provide micro-level controllability of MAC [78, 29]. Some of the benefits of micro-level controllability are:

  i) Predictability: Accurate performance predictability is possible when channel access is well controlled. Predictability is useful for network selection, network arrangement, and applications that require predictable service.

  ii) Differentiation: Service differentiation is important when providing Quality of Services (QoS) to clients. By controlling what is transpiring in the network, a fine-grained service differentiation can be provided.

  iii) Efficiency: WiFi networks have poor channel efficiency in high-density deployments and hidden terminal scenarios due to severe collisions. By controlling who access the channel currently, higher efficiencies can be achieved by avoiding collisions.

Along with different enhancements, these protocols and algorithms also require the different complexity of changes to the WiFi infrastructure. Some algorithms require only changes in APs [45, 65, 36]. Some algorithms require changes in APs and stations (STAs) [21, 51]. In addition to the evolution in APs and STAs, some algorithms require a centralized controller [78, 64]. Some algorithms even require extra time synchronization [29]. Due to the huge number of WiFi devices, the cost of updating/replacing all devices to new

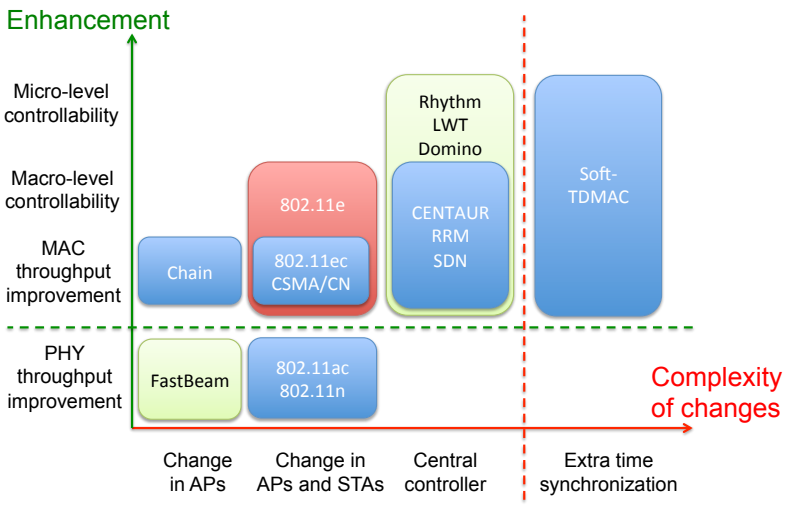


Figure 1: Spectrum of WiFi algorithms and protocols

protocols can be very high. This is especially a concern for enterprises, who have invested significantly in legacy WiFi infrastructure. Thus, the complexity of changes is also an important consideration when adopting new protocols and algorithms.

Fig. 1 shows a spectrum of enhancement and complexity of changes in protocols and algorithms for next generation WiFi networks. While there is a huge possibility of locating protocols and algorithms in the spectrum, without considering the "extra time synchronization" portion[1], we would like to focus on the two corners of the spectrum. One acts as a transition solution for updating legacy WiFi networks, and the other considers future-proofing networks and enable micro-level controllability to support service differentiation and even performance prediction. Our work comprises three components considering the two portions:

- Improve PHY layer throughput with only changes in APs: First, we propose algorithms that improve the performance in the PHY layer with only changes in APs. Recently, smart antenna technology provides tremendous performance enhancement in the PHY layer of WiFi. While most smart antenna techniques require changes in

[1]The requirement of extra time synchronization adds considerable burden on the network operations and thus is still not very practical in most applications of WiFi.

both APs and STAs, it is possible to apply one technique, beamforming, with only changes in APs. We propose **FastBeam** [65], a set of algorithms that can bring beamforming benefits to legacy nodes only by upgrading APs. With only requirements of updates in APs, algorithms can be easily deployed. These algorithms provide a temporary transition solution for updating the legacy WiFi networks with a huge number of nodes. The main challenge in FastBeam is in efficient estimation of channel state information and the tradeoff between accuracy and time complexity. We implemented algorithms with different level of tradeoff between accuracy and time complexity and applied them according to the current dynamic level of channel state information.

- Achieve micro-level controllability with a central controller: Second, we consider future-proofing networks with a central controller. Since WiFi has broad applications, the service requirements are diverse. Thus, the goal is to enable micro-level controllability to support service differentiation and even performance prediction for various applications. However, the MAC of WiFi is random, hard to predict, and hard to control. We focus on how to **enable the controllability of WiFi networks without compromising their scalability** when a central controller is available. We introduce a MAC protocol called **Rhythm** [64, 63], which transfers the control of WiFi networks into centralized scheduling. Rhythm is subject to the constraints of (i) no fine-grained time synchronization, (ii) no additional hardware requirement; and with the properties of (i) low protocol overhead, (ii) work conservation in the presence of non-backlogged nodes, (iii) robustness to partial connectivity scenarios. Rhythm provides all nodes in WiFi networks a **target schedule** that has been determined by a central controller. The target schedule represents the control decision for the WiFi networks. The nodes then operate in a **purely distributed fashion** to follow the target schedule. That is, once the nodes get the target schedule, they transmit in the order indicated by the target schedule without further help from the central entity.

We refer to such a network behavior as "**scheduled WiFi**". The primary challenge of Rhythm is in dealing with random traffic without gathering queue information from STAs, and in handling hidden terminal problems. To handle random traffic, we designed a schedule shrinking and insertion algorithm to efficiently remove nodes from the schedule when they do not have packets to transmit, and put them back once they become active. To handle hidden terminal, we proposed a Clique and Bridge structure which groups nodes in communication range into Cliques, and pass control information through nodes that act as Bridges between different Cliques.

- Achieve micro-level controllability considering backward compatibility: Finally, we propose **LWT** [62], which considers the backward compatibility issues of scheduled WiFi. Similar to Rhythm, LWT achieves scheduled WiFi with little overhead. While Rhythm assumes an ideal environment without legacy nodes, LWT provides backward compatibility with legacy nodes by allowing for small deviations from target schedule intermittently. The primary challenge of LWT is proving backward compatibility in the presence of random traffic and hidden terminals. To handle random traffic, we propose an algorithm that efficiently interleaves the LWT mode and DCF mode in small time scales. If a scheduled node does not transmit, the operation mode directly switches to DCF mode, and all nodes can compete to transmit. To handle hidden terminals, we propose a novel mechanism, called **Switch**, which uses specially designed signal pulse to robustly deliver control information, without affecting the ongoing transmission significantly. By making transmitter and receiver switch between transmit and receive mode in a dedicated designed pattern, Switch provides a way to utilize overlay control channels better compared to existing works [27, 43, 71]. In addition to the details of how Switch is used in LWT, we further explored how Switch can be used in other situations, such as i) extending the range of carrier sense, ii) early collision termination, and iii) improving the efficiency of WiFi backoff mechanism.

The core of our research is developing algorithms. The research philosophy that has guided us in our research is to support our algorithms with sound theoretical analysis and real-life experiment evaluations after the algorithms have been developed. Thus, the algorithm designs have been strongly guided by both theory and practical implementations. In all these research, we have written theoretical analysis and proof of properties, implemented experimental test beds, and carried out simulation evaluations.

The rest of this thesis is organized as follows: Chapter II gives a literature survey, Chapter III presents FastBeam, Chapter IV describes Rhythm, Chapter V introduces LWT, Chapter VI shows Switch, and Chapter VII concludes our findings.

CHAPTER II

LITERATURE SURVEY

WiFi is a wireless communication technology that connects devices to local area networks. Due to its high-data-rate achievability, cheap deployability, and near-universal availability, WiFi has been widely adopted in different environments to provide various services. Various protocols and algorithms have been proposed to enhance different abilities of WiFi in both the Medium Access Control (MAC) and physical layer (PHY) layers to support the ubiquitous applications of WiFi. In this chapter, we introduce protocols and algorithms related to smart antenna technologies and controllability enhancement for WiFi networks and compare them to the proposed algorithms of this thesis.

2.1 Smart antenna technologies in the PHY layer

Several research works have applied smart antenna technologies to WiFi networks, considering indoor multipath environment or mobile clients.

2.1.1 Smart antenna technologies for indoor wireless LANs

MIMO and multi-user MIMO [20, 21] can provide tremendous throughput improvement. However, these techniques require changes in both APs and STAs, unlike FastBeam, which requires changes only to APs. [47], [48], [49], and [50] consider the usage of directional antennas in indoor environments. While directional antenna could have benefits compared to Omni antennas, the beamforming technology used by FastBeam has a larger potential to improve performance under dedicated setting in multipath-rich indoor environments.

[46] and [45] consider the use of beamforming technology in indoor environments. [46] presents the design and implementation of the first indoor WLAN beamforming system. It identifies and addresses several challenges with beamforming that are often ignored in

theoretical works. However, the use of CSI in the proposed solution will require clients to equip specialized hardware. [45] (SimpleBeam) provides an algorithm that enables an off-the-shelf client to gain the benefits of beamforming without hardware modifications. However, the proposed algorithm (as will be shown in chapter III) is not suitable for an environment with dynamic channel conditions.

2.1.2 Smart antenna technologies for mobile clients

[52] uses directional antennas to improve the performance of WiFi links between roadside access points and a moving vehicle equipped with directional antennas. A framework *MobiSteer* is proposed to select the best access point and beam combination that maximizes the throughput. [57] implements a vehicular communication system that uses multi-lobe beam pattern switching on a smart antenna to improve the uplink connectivity. Both [52] and [57] focuses on outdoor environments. Multipath-rich indoor environments are more complicated to improve performance. [74] demonstrates that beamforming is already feasible on mobile devices. It proposes *BeamAdapt* which allows each client to identify the optimal number of active antennas that could achieve power efficiency while maintaining a required throughput. Unlike FastBeam, which does not require CSI, [55] uses CSI to implement beamforming and improve energy efficiency.

2.2 Controllability Enhancement for WiFi networks

Protocols and mechanisms have been proposed to increase the controllability of WiFi networks in the MAC layer.

2.2.1 Centralized control of WiFi networks

Many solutions have been proposed for centralized control of WiFi networks. Interoperability between AP and WiFi nodes has been proposed to perform RRM across APs from multiple vendors (IEEE 802.11k [17]). WiFi controller moves functionalities like RRM, mobility, wireless policies, QoS and authentication services to the central controller [1].

Software Defined Networking (SDN) presents a logically centralized software to separate network control logic from physical switches and routers. SDN is another move towards centralizing control of the network [12]. RFC 5412 [19], a draft of Light Weight Access Point Protocol (LWAPP) has been submitted to the IETF for communication between AP and the central controller. The central controller makes network-wide decisions like Radio Resource Management (RRM), to provide dynamic channel assignment, dynamic transmit power control, load balancing, authentication, and membership services. Wireless vendors like Cisco, Ruckus have proposed software and mechanisms (CAPWAP [18]) to manage centralized WLAN. CloudMAC [69] offers an OpenFlow-based MAC protocol in which all the processing is performed in the cloud, and the APs use DCF to contend and transmit a packet. All these schemes, while still operating with the unpredictable and hard-to-control DCF, can only provide macro-level controllability.

2.2.2 Centralized MAC protocols

Centralized MAC protocols, such as Soft-TDMAC [29] and PCF [20], can provide MAC level controllability. However, PCF introduces impractical protocol overhead when the number of non-backlogged nodes is large. It also generates repeated collisions when multiple APs are operating in the same vicinity, and thus is not a good choice in most practical situations. Soft-TDMAC makes nodes transmit control packets to achieve time synchronization. Thus, scheduling decisions can directly be applied to nodes (just like TDMA). However, it requires tight time synchronization for good efficiency, and this adds to the burden of network operations[1]. It also requires queue status of nodes for scheduling. Thus, it either needs to assume that nodes always transmit or incurs significant overhead in collecting queue status from all nodes.

[1]Note that Rhythm and LWT utilize overheard transmissions to synchronize scheduling/contention; thus, Rhythm and LWT are not affected by clock drifts. This is very different from the traditional synchronized CSMA or TDMA, which is influenced by clock drifts.

2.2.3 Hybrid MAC protocols

Hybrid MAC protocols have been proposed to increase the controllability of WiFi networks and to avoid the drawback of centralized MAC protocol. CENTAUR [66] utilizes the centralized WiFi architecture for enhanced performance of WiFi. It solves hidden terminal problems by separately scheduling conflicting downlink transmissions. However, it only gives control to downlink traffic, so it does not avoid hidden terminal problems generated by uplink traffic. It also improves performance, but only in cases with hidden/exposed terminal problems. On the other hand, Rhythm and LWT can improve performance in cases with or without hidden/exposed terminal problems, and avoid hidden terminal problems in both uplink and downlink. Another protocol, Chain [75] broadcasts order coordination, which indicates a transmission order of WiFi nodes, to achieve high channel utilization. However, it improves performance only when the traffic load is high, while Rhythm and LWT can maintain good performance for different traffic loads. Also, Chain neither addresses hidden terminal problems nor improves downlink throughput. Similar to Rhythm and LWT, Domino [78] utilizes relative scheduling to avoid overhead from tight time synchronization. While Domino provides an efficient way of collecting the queue status from all nodes, Rhythm and LWT provides distributed adjustment mechanisms without having to obtain the queue status. However, as will be indicated in Chapter IV and V, Domino generates extra protocol overhead and unfairness in certain scenarios. Besides, it requires the use of particular addresses derived from Gold codes, which either limits the number of nodes (127 in its evaluation) in a collision domain or increases overhead by using larger signatures for addressing.

CHAPTER III

FASTBEAM: BEAMFORMING STRATEGIES FOR PERFORMANCE IMPROVEMENT ON LEGACY CLIENTS

3.1 Introduction

Transmit beamforming enables a transmitter to direct signals at an intended receiver; thus, using beamforming can improve the signal quality and hence the data throughput and range for a wireless link [55]. Even in indoor environments, where directional antennas with fixed radiation patterns fail to provide performance improvements due to multipath fading, beamforming can still deliver substantial benefits through the adjustment of the relative weights and phases of the different elements of an antenna array. By default, beamforming requires the complete Channel State Information (CSI). However, collecting such information at the receiver requires specialized hardware, which is not available on legacy WiFi clients. Recent techniques [45] have demonstrated that optimal beamforming can be achieved while relying purely on the Receive Signal Strength Indicator (RSSI), a parameter that is readily and easily available in legacy WiFi clients. However, the existing techniques for RSSI-based beamforming do not reach achievable performance in *time varying channels*, and can, in fact, perform worse than Omni-directional antennas under some scenarios. We first carry out experiments to evaluate the performance degradation of RSSI-based beamforming in time varying channels. We then explore algorithms to perform *fast beamforming* when relying only on RSSI measurements. Fast beamforming allows for a faster continuous adaptation of the antenna weights for environments where the *channel is time varying* because of client mobility, client orientation changes, and environmental changes (such as changes in positions of objects/people).

We thus present FastBeam, a suite of strategies that in tandem enables practical beamforming in time-varying environments. FastBeam uses only RSSI measurements for performing beamforming and consists of both optimality preserving techniques and heuristic approaches that are selectively applied depending on the rate of change of the channel. We implement $FastBeam$ on a Phocus Array System ([14]) with eight antennas and use experimental evaluations to study its performance in indoor environments. We show that the time complexity for RSSI-based beamforming can be reduced by an average of 50% and by as much as 75% compared to existing approaches ([45]). Such a reduction in the time complexity has a significant impact on the throughput performance under several conditions: we show that the throughput performance of $FastBeam$ compared to the existing work is 1.4x better on average, and up to 1.8x better in the best case.

3.2 Background of RSSI Based Beamforming

3.2.1 Beamforming with RSSI

Beamforming techniques can deliver considerable improvements over Omni-directional communication in both outdoor and indoor environments. Specifically, in indoor environments, while simple directional antenna with fixed radiation patterns suffers from multipath fading and scattering, adaptive beamforming still holds tremendous promise. A recent advancement in beamforming is a partial-CSI approach called *RSSI based beamforming* [45], which only uses RSSI measurements sent as feedback from the client to make beamforming decisions. While it is provably optimal (same performance as traditional beamforming under static channel conditions), the most attractive property of RSSI-based beamforming is that it does not require the receiver to be equipped with an antenna array or specialized hardware to provide benefits. This makes the technique attractive to cater to off-the-shelf wireless clients.

3.2.2 Algorithm for RSSI-based Beamforming

Assuming that there are n antenna elements at the transmitter, and a single antenna at the receiver, the consequent Multiple Input Single Output (MISO) channel can be represented as follows:

$$y = \mathbf{hw}s + z, \tag{1}$$

where y is the received signal, z is the additive White Gaussian noise, the $1 \times n$ vector of complex number $\mathbf{h} = [h_0, h_1, \cdots h_{n-1}]$ is the respective channel gains between each transmitter antenna and the receiver antenna, and the $n \times 1$ vector of complex numbers $\mathbf{w} = [w_0, w_1, \cdots w_{n-1}]^T$ is a beamformer that translates the transmit symbol s to the transmitted signals $\mathbf{x} = \mathbf{w}s$. The optimal setting of beamformer $\mathbf{w}$ is to let each beamformer weight w_i become the complex conjugate of h_i, so that the signals from each channel combine coherently and reinforce each other at the receiver.

RSSI based beamforming does channel estimation and calculates the beamformer $\mathbf{w}$ according to the RSSI values that are sent as feedback from Rx. The core idea is to use both singly and tandemly activated antennas to determine the channel magnitude and phase difference between the antennas. When a single antenna is activated, the respective received powers $P_i = |h_i|^2$ (assuming the Tx power is unity) are obtained from the RSSI value sent as feedback from Rx. When antennas i and j are tandemly activated (with antenna phase difference ρ_{ij} set to 0), the received powers $P_{ij} = |h_i + h_j|^2$ are obtained. P_{ij} can be written as:

$$P_{ij} = P_i + P_j + 2\sqrt{P_i P_j}\cos(\theta_{ij}), \tag{2}$$

where θ_{ij} is the channel phase difference between h_i and h_j. By rewriting equation (2), we can get θ_{ij} using P_i, P_j, and P_{ij}:

$$\cos\theta_{ij} = (\frac{P_{ij} - P_i - P_j}{2\sqrt{P_i P_j}}), \tag{3}$$

$$\phi_{ij} = \cos^{-1}(\frac{P_{ij} - P_i - P_j}{2\sqrt{P_i P_j}}). \tag{4}$$

Since $|\cos\theta| = |\cos(-\theta)| = |\cos(\pi - \theta)| = |\cos(-\pi + \theta)|$, there are 4 possible cases: $\theta_{ij} = \phi_{ij}$, $\theta_{ij} = -\phi_{ij}$, $\theta_{ij} = \pi - \phi_{ij}$, or $\theta_{ij} = -\pi + \phi_{ij}$. In $SimpleBeam$ ([45]), this ambiguity is solved through another 4 sets of tandemly activated antennas. Let $P_{ij(\rho_{ij})}$ represent the received power of tandemly activated antennas i and j with antenna phase difference set to ρ_{ij} ($P_{ij(\rho_{ij})} = P_i + P_j + 2\sqrt{P_i P_j}\cos(\theta_{ij} + \rho_{ij})$). $P_{ij(\phi_{ij})}$, $P_{ij(-\phi_{ij})}$, $P_{ij(\pi-\phi_{ij})}$, and $P_{ij(-\pi+\phi_{ij})}$ are measured, and the antenna phase setting that corresponds to the maximum received power indicates the value of $-\theta_{ij}$. [1]

With the $|h_i|$, θ_{i0}, and the channel phase difference between h_i and h_0 computed for each antenna i, the beamformer weights are set as $w_i = \frac{|h_i|}{\sqrt{\Sigma_{l=0}^{n-1}|h_l|^2}} e^{-\mathbf{j}\theta_{i0}}$ for $i > 0$ and $\mathbf{j} = \sqrt{(-1)}$, and $w_0 = \frac{|h_0|}{\sqrt{\Sigma_{l=0}^{n-1}|h_l|^2}}$.

3.2.2.1 Optimality of RSSI-based Beamforming

RSSI based beamforming is provably optimal (same performance as traditional beamforming). According to equation (1), the optimal beamforming setting should be $\mathbf{w} = \frac{1}{\sqrt{\Sigma_{l=0}^{n-1}|h_l|^2}}\mathbf{h}^*$. In our setting, the phase of w_i is set to $-\theta_{i0}$, which is $-\arg(h_i) + \arg(h_0)$ rather than $-\arg(h_i)$; the phase of w_0 is set to 0 rather than $-\arg(h_0)$. This alternative setting only introduces a phase shift of $\arg(h_0)$ in $\mathbf{w}$, and thus will not affect the received signal power. Thus, this alternative setting also achieves the same optimal performance, as is proved in [45].

[1] In fact, since the exact value of $\cos\theta_{ij}$ is known, two sets of tandemly activated antennas is sufficient to get θ_{ij}. However, since time-complexity is not a concern in [45], it uses four sets of tandemly activated antennas. In section 3.4, we will provide an even more efficient algorithm using *only one* set of tandemly activated antennas.

3.2.2.2 *Practical Advantages of RSSI-based Beamforming*

RSSI-based beamforming is optimal, and also has several practical benefits:

(i) Full CSI approaches require specialized clients with the capability to measure the amplitude and phase of the received signals. On the other hand, RSSI-based beamforming can be implemented with off-the-shelf clients.

(ii) Oscillator related hardware synchronization impairments tend to corrupt the estimated CSI. RSSI-based beamforming, on the other hand, solves the synchronization impairments by using the differential phase mechanism (using $-(\arg(h_i) - \arg(h_0))$ rather than $-\arg(h_i)$) [46],

(iii) Full CSI based beamforming approaches incur considerable overheads in the feedback sent to the transmitter. RSSI based beamforming incurs small overheads as only RSSI values need to be communicated.

3.3 Problem Motivation

We use $SimpleBeam$ as the representative solution for RSSI-based beamforming. Although $SimpleBeam$ is optimal and has better performance under static channel conditions[45] demonstratively, its performance under varying channel conditions (when $\mathbf{h}$ varies) has not been studied before.

We now present results from experiments that explore the performance of $SimpleBeam$ when $\mathbf{h}$ varies. The experimental setting is described in Sec. 3.6. We consider mobile environments: (i) when the client orientation changes over time (Rotation), (ii) when there is human movement interfering with the line of sight (LOS Blocking) periodically, and (iii) when the client is moving (Mobility). Under these environments, the variation of $\mathbf{h}$ becomes larger due to the large change in multipath fading conditions. Table 1 shows the degradation of throughput performance of $SimpleBeam$ in comparison to that in a static environment (when $\mathbf{h}$ is more static). Figure 2 shows that under certain conditions,

$SimpleBeam$ can perform even worse than Omni-directional antennas (Omni). These results show that $SimpleBeam$ doesn't perform well when the channel is time varying.

We contend that the underlying reason for the poor performance under channel varying conditions is the high time complexity of $SimpleBeam$. With n antenna elements, $SimpleBeam$ relies on n single antenna channel estimations, $n-1$ tandem antenna channel estimations, and an additional $4(n-1)$ tandem antenna estimations to solve the ambiguity due to the $\cos^{-1}$ function. Thus, the time complexity is $(6n-5)$ChannelEstTime, where ChannelEstTime is the time of channel estimation. To gain optimal performance, $SimpleBeam$ needs to adjust the beamformer $\mathbf{w}$ according to the channel gains $\mathbf{h}$. When the rate of change of $\mathbf{h}$ becomes faster than the convergence time of $SimpleBeam$, the estimation of $\mathbf{h}$ will be erroneous, which causes significant performance degradation.

In Figure 3, we show an analysis of the phase fluctuation of $\mathbf{h}$ observed in two experimental setups. We see that the channel phase variation can be as large as 100 degrees within a time span of tens of seconds with LOS blocking every 5 seconds, while it is only 30 degrees in a static environment. If the algorithm starts when $t = 50$ and finishes when $t = 70$, there will be a phase error $\theta_e = 100$ with LOS blocking every 5 seconds. The maximum θ_e is 30 degree in a static environment. Using equation (2) and assuming $P_i = P_j$, we can estimate the degradation percentage with phase error θ_e using equation:

$$\frac{P_{\theta_e} - P_{opt}}{P_{opt}} = \frac{P_i + P_j + 2\sqrt{P_i P_j}\cos\theta_e - P_i - P_j - 2\sqrt{P_i P_j}}{P_i + P_j + 2\sqrt{P_i P_j}} = \frac{2P_i(\cos\theta_e - 1)}{4P_i}$$
$$= \frac{1}{2}(\cos\theta_e - 1).$$

Accordingly, a 30 degrees phase error will result in only -6.69% degradation, while a 100 degrees phase error will result in -58.68%. Thus, the performance degradation will be significant if $\mathbf{h}$ differs a lot by the time the computation results are applied.

With n antenna elements, the time complexity of $SimpleBeam$ is $(6n-5)$ChannelEstTime, where ChannelEstTime is the time of channel estimation. In our experimental setup with

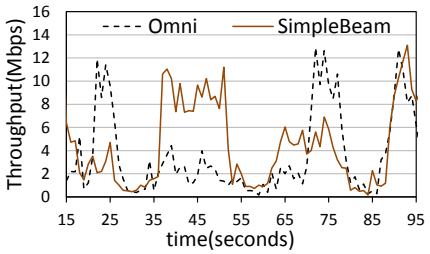


Figure 2: Comparison of $SimpleBeam$ and Omni when clients are moving at a speed of 1.5m/s

Table 1: Performance degradation of $SimpleBeam$ in a mobile environment

Rotation		LOS blocking		Mobility	
degree/s	%	Freq(1/s)	%	m/s	%
180	-64.84	1/10	-42.07	0.5	-37.28
360	-88.24	1/5	-71.44	1.5	-58.49

eight antenna elements, $SimpleBeam$ takes 22 to 27 seconds to perform its channel estimation (an implementation of the mechanism described in [45]). The high time complexity of $SimpleBeam$ thus is a drawback as channel gains $\mathbf{h}$ are likely to be different by the time algorithm finishes computing.

3.4 FastBeam Design

In this section, we present the key design elements of $FastBeam$, a set of four techniques that addresses the drawbacks discussed in Section 3.3 by reducing the time complexity. The first technique is optimality preserving, while the other three methods are heuristic strategies toward reducing time complexity.

3.4.1 FewerPhases (optimality preserving)

Based on the mathematical analysis, we argue that the time complexity of $SimpleBeam$ could be reduced while still preserving its optimality. In $SimpleBeam$, four channel estimations are carried out for each channel phase difference θ_{ij} to solve the ambiguity due to the $\cos^{-1}$ function. This results in a time complexity of $4(n-1)\mathsf{ChannelEstTime}$ when there are n antennas. Here we prove that the ambiguity can be resolved using only one estimation for each channel phase difference θ_{ij}. We implement this concept in the algorithm

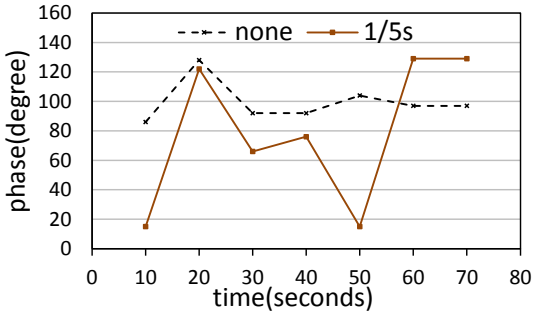


Figure 3: Fluctuation of channel phase when LOS blocking exists

$FewerPhases$.

3.4.1.1 *Description*

Instead of carrying out 4 channel estimations to solve the ambiguity, $FewerPhases$ uses only 1 tandemly activated channel estimation to find θ_{ij} and sets $\mathbf{w}$ accordingly. Let ρ_{ij} be the antenna phase difference of tandemly activated antenna i and j. Consider two cases:

Case 1 : If $\phi_{ij} \leq \frac{\pi}{2}$, set $\rho_{ij} = \phi_{ij}$.

Case 2 : If $\phi_{ij} > \frac{\pi}{2}$, set $\rho_{ij} = \pi - \phi_{ij}$.

If $P_{ij(\rho_{ij})} \leq P_{ij}$, $\theta_{ij} = \phi_{ij}$; else, $\theta_{ij} = -\phi_{ij}$.

3.4.1.2 *Proof of Optimality*

According to [45], θ_{ij} could be ϕ_{ij}, $-\phi_{ij}$, $\pi - \phi_{ij}$, or $-\pi + \phi_{ij}$. However, since it is possible to exactly know whether the $\cos(\theta_{ij})$ is positive or negative by calculating whether $(P_{ij} - P_i - P_j)$ is positive or negative, there are only two possibilities: $\theta_{ij} = \phi_{ij}$ or $\theta_{ij} = -\phi_{ij}$.

In Case 1, the received power will be as follows:

$$P_{ij(\rho_{ij})} = P_i + P_j + 2\sqrt{P_i P_j} cos(\theta_{ij} + \phi_{ij}). \tag{5}$$

i) If $\theta_{ij} = \phi_{ij}$, we will have $P_{ij(\rho_{ij})} = P_i + P_j + 2\sqrt{P_i P_j} cos(\theta_{ij} + \theta_{ij})$, and since $\cos(2\theta) \leq \cos(\theta)$ when $|\theta| \leq \frac{\pi}{2}$, compared to equation 2, we have $P_{ij(\rho_{ij})} \leq P_{ij}$.

ii) On the other hand, if $\theta_{ij} = -\phi_{ij}$, we will have $P_{ij(\rho_{ij})} = P_i + P_j + 2\sqrt{P_i P_j} cos(\theta_{ij} - \theta_{ij})$, and since $\cos(0) \geq \cos(\theta)$, we have $P_{ij(\rho_{ij})} \geq P_{ij}$.

In Case 2, following the similar deduction in Case 1, we have $P_{ij(\rho_{ij})} \leq P_{ij}$ when $\theta_{ij} = \phi_{ij}$, and $P_{ij(\rho_{ij})} \geq P_{ij}$ when $\theta_{ij} = -\phi_{ij}$.

3.4.1.3 ***Reduced Complexity***

The time complexity of dealing with ambiguity in $FewerPhases$ is thus reduced to $(n-1)$ChannelEstTime. Thus, the total time complexity for $FastBeam$ is reduced to $(3n-2)$ChannelEstTime.

3.4.2 FewerAntennas (heuristic)

When measuring $\mathbf{h}$, a large diversity in the magnitude of channel gains across antennas can be observed from time to time. While the antennas corresponding to the channels with poor magnitude do not help a lot in performance improvement, the time complexity of calculating the optimal beamformer weight of each channel is the same. Thus, it is reasonable to stop using the antennas corresponding to the channels with a poor magnitude and hence save on the time complexity. $FewerAntennas$ adaptively reduces the number of antennas used, and hence reduces the time complexity.

3.4.2.1 ***Description***

$FewerAntennas$ first performs single channel estimation and gets the P_i for each antenna, and sorts them according to their power. Without loss of generality, we assume that $P_0 \geq P_1 \geq P_2 \cdots \geq P_n$. Then $FewerAntennas$ determines the number of antennas to use, n_{ant}, using the following equation:

$$n_{ant} = \arg_k \min(\Sigma_{i=0}^{i=k} \sqrt{P_i} \geq \alpha \Sigma_{i=0}^{i=n} \sqrt{P_i}), \tag{6}$$

where $\arg_k \min(Statement)$ returns the smallest k that satisfy the *Statement*, and $\alpha < 1$ is a threshold which guarantees that the received power is at least a fraction α of the total

power.

After determining the number of in-use antennas n_{ant}, the rest of the $FastBeam$ algorithm is same as that of $FewerPhases$, but with a reduced set of antennas.

3.4.2.2 ***Justification***

Although using fewer antennas will decrease the maximum possible received power, the reduction in time complexity can reduce the estimation error for $\mathbf{h}$, which results in better performance. As mentioned in section 3.3, using equation (2) and assuming $P_i = P_j$, we can estimate the degradation percentage with phase error θ_e using equation $\frac{P_{\theta_e} - P_{opt}}{P_{opt}} = \frac{1}{2}(\cos\theta_e - 1)$. Thus, $\theta_e = 30$ will result in only -6.69% degradation from optimal received power, while $\theta_e = 100$ will result in -58.68%. Therefore, there is a large opportunity to decrease the degradation by sacrificing the maximum possible received power and reducing the time complexity. For example, assuming $\alpha = 0.8$, and due to the reduction in time complexity, we could have $\theta_e \leq 30$ degrees when using $FewerAntennas$. The degradation of $FewerAntennas$ is at most $\alpha \times (1 - 6.69\%) - 1 = -25.35\%$. On the other hand, $SimpleBeam$ could have $\theta_e = 100$ degrees and result in a -58.68% degradation. In this example, the received power of $FewerAntennas$ is 1.8x better than $SimpleBeam$.

3.4.2.3 ***Reduced Complexity***

The time complexity of tandemly activation and ambiguity resolution are both reduced to $(n_{ant} - 1)$ChannelEstTime. Thus, the total time complexity is reduced to $(n + 2n_{ant} - 2)$ChannelEstTime.

3.4.3 ZeroPhase (heuristic)

Although $FewerAntenna$ decreases the time complexity by a certain amount, to deal with environments having a high variance of channel conditions, it is desirable to have an algorithm that has even less time complexity, at the cost of accuracy. $ZeroPhase$ is a heuristic algorithm that attempts to construct the beam pattern without even computing the exact

channel phase θ_{ij}. It simply set up the beamformer in a way that the signal on each antenna are constructive rather than destructive to the signal of the reference antenna (i.e. antenna 0).

3.4.3.1 ***Description***

$ZeroPhase$ first carries out the singly and tandemly activated channel estimations to get P_i and P_{i0} for each antenna. Then, for each antenna i, $i \neq 0$, if $P_i + P_0 > P_{i0}$, the phase of w_i is set as π; else, the phase of w_i is set as 0.

3.4.3.2 ***Justification***

Since $P_{i0} = P_i + P_0 + 2\sqrt{P_i P_0}cos(\theta_{i0})$, if $|\theta_{i0}| > \frac{\pi}{2}$, we will have $P_i + P_0 > P_{i0}$. By setting the phase of w_i as π, the received power of the two antenna will become $P_{i0} = P_i + P_0 + 2\sqrt{P_i P_0}cos(\theta_{i0} + \pi)$, and since $\cos(\theta_{i0} + \pi) > 0$, which will make $P_i + P_0 < P_{i0}$, the destructive effect of the two signals becomes constructive. On the other hand, if we already have $P_i + P_0 < P_{i0}$, we will simply set the phase of w_i as 0 to keep it constructive.

Further, we use the equation: $\frac{1}{2}(\cos\theta - 1)$ from section 3.3 to estimate the degradation of $ZeroPhase$. Since $ZeroPhase$ always control the signal to be constructive, the θ_e is at most 90 degree: the degradation is at most $\frac{1}{2}(\cos\frac{\pi}{2} - 1) = -50.00\%$; compared to the possible -58.68% degradation in $SimpleBeam$, the received power of $ZeroPhase$ is 1.2x better in the worst case.

3.4.3.3 ***Reduced Complexity***

Since $ZeroPhase$ only needs to carry out the singly and tandemly activated channel estimations, the total time complexity is reduced to $(2n-1)\mathsf{ChannelEstTime}$.

3.4.4 FingerPrint (heuristic)

Beyond the strategies discussed so far, with historical information about the environment, the time complexity can be reduced even further. $FingerPrint$ is a heuristic algorithm

that uses the pre-measured, recorded (fingerprint $\mathbf{f_k}$, beamformer $\mathbf{w_k}$) pairs to set up the beamformer $\mathbf{w}$. The fingerprint $\mathbf{f}$ is a sequence of received power values that reflect the characteristics of the channel gain $\mathbf{h}$. The algorithm measures the fingerprint $\mathbf{f}$ of the current channel gain $\mathbf{h}$, compares it with previously recorded fingerprints $\mathbf{f_k}$, and then uses the corresponding recorded beamformer $\mathbf{w_k}$ if there is a match in the fingerprint.

3.4.4.1 ***Description***

$FingerPrint$ will first measure the fingerprint $\mathbf{f} = (P_0, P_{01}, P_{02}, \cdots, P_{0n})$ of the current channel gain $\mathbf{h}$, and search among its data base of fingerprints ($\mathbf{f_k}$, $\mathbf{w_k}$). If there is a match, i.e. there is a $\mathbf{f_k}$ having similar characteristic as $\mathbf{f}$, $FingerPrint$ sets the beamformer as $\mathbf{w_k}$; otherwise the algorithm will run $FewerPhase$ and record the resulting fingerprint and beamformer pattern ($\mathbf{f}$, $\mathbf{w}$). The matching function is: $|\mathbf{f} - \mathbf{f_k}|$. If the value of matching function is smaller than a threshold β, there is a match.

3.4.4.2 ***Justification***

Each channel gain $\mathbf{h}$ will have its own fingerprint $\mathbf{f}$. If $\mathbf{h} \neq \mathbf{h}'$, the probability that $\mathbf{f} = \mathbf{f}'$ is very small. Since the fingerprint $\mathbf{f}$ contains the power level of reference antenna $P_0 = |h_0|^2$ and the $P_{0i} = |h_0 + h_i|^2$, $\mathbf{f}$ captures the channel phase difference θ_{0i} between h_0 and h_i. Assume that there are k_p possible level of channel power gain $|h_i|$ and 360 possible degree of channel phase $\arg(h_i)$. That is, there are $k_p \times 360$ possible value of h_i. According to equation (3), given values of P'_{0i}, $P_0 = P'_0$, and $P_i = |h_i|^2$, there are only two possible values for θ_{0i} that make $P_{0i} = P'_{0i}$. Assuming uniform distribution of channel phase $\arg(h_i)$ and channel power gain $|h_i|$, we will have:

$$Pr(P_0 = P'_0 | P'_0) = \frac{360}{k_p \times 360} = \frac{1}{k_p},$$
$$Pr(P_{01} = P'_{01} | P'_{01}, P_0 = P'_0, h_1 \neq h'_1) = \frac{2 \times k_p - 1}{360 \times k_p} \cong \frac{1}{180}.$$

Thus,

$$Pr((P_0, P_{01}) = (P'_0, P'_{01}) | P'_{01}, P'_0, (h_0, h_1) \neq (h'_0, h'_1)) \cong \frac{1}{180 \times k_p},$$

Algorithm 1 FastBeam algorithm

INPUT:

V = variable indicating the level of RSSI variance at client. The value of V could be "Low", "Medium", or "High".

ALGORITHM:

```
1: for each update cycle or request for update from client do
2:     f = Fingerprint_Measurement();
3:     w = Recorded_Pattern.match(f);
4:     if w != NULL then
5:         jump to line 14;
6:     else if V == "Low" then
7:         w = FewerPhase(f);
8:         Recorded_Pattern.record(f,w);
9:     else if V == "Medium" then
10:        w = FewerAntennas(f);
11:    else if V == "High" then
12:        w = ZeroPhase(f);
13:    end if
14:    set up w;
15: end for
```

and

$$Pr(\mathbf{f} = \mathbf{f}'|\mathbf{f}', \mathbf{h} \neq \mathbf{h}') \cong \frac{1}{180^{(n-1)} \times k_p}.$$

If $n = 8$, $k_p = 10$, $Pr(\mathbf{f} = \mathbf{f}'|\mathbf{f}', \mathbf{h} \neq \mathbf{h}') < 10^{-15}$.

Therefore, if we measured the current fingerprint $\mathbf{f}$ and infer that $\mathbf{f} = \mathbf{f}'$, there is a high probability that $\mathbf{h} = \mathbf{h}'$. If the pair ($\mathbf{f}'$, $\mathbf{w}'$) was recorded earlier, the beamformer $\mathbf{w}'$ can be applied. Since there always exists a little variance in the measurement values of power, $FingerPrint$ does not strictly require that $\mathbf{f} = \mathbf{f_k}$. As long as the difference $|\mathbf{f} - \mathbf{f_k}|$ is smaller than a preset threshold, $FingerPrint$ infers them to be equal.

3.4.4.3 ***Reduced Complexity***

Since $FingerPrint$ only needs to carry out one single activation and the tandemly activated channel estimation, the total time complexity is reduced to nChannelEstTime if there is a match. If there is no match, the time complexity will be the same as $FewerPhase$.

3.5 FastBeam Solution

We now present the details of $FastBeam$ solution composed of the four algorithms described in Section 3.4. We consider one AP equipped with multiple antennas and one off-the-shelf client with an Omni antenna. The pseudo code for $FastBeam$ is presented in Algorithm 1.

$FastBeam$ is an algorithm that requires client participation. However, the client participation is minimal and is purely in software. Since it is necessary to carry out smaller time complexity algorithms when channels change rapidly, $FastBeam$ selects among the four algorithms based on the variance of RSSI measured by the client. Based on the RSSI records over time, the client reports the rate of change of its RSSI values using a three-level indicator: "High," "Medium," and "Low." The level of variance V is sent to the AP periodically or when the client wants to trigger an update of the beamformer $\mathbf{w}$ if the RSSI is smaller than a threshold.

The AP performs the $FastBeam$ algorithm periodically or when it receives an explicit request from the client. Since we always have excellent performance when there is a match in $FingerPrint$, here we design and implement $FastBeam$ to use the recorded beamformer $\mathbf{w}$ whenever there is a match. A more careful design could carry out an RSSI measurement for the matched beamformer $\mathbf{w}$ to see if the performance is satisfiable, and decide not to use the matched beamformer $\mathbf{w}$ if the measured RSSI is low.

Algorithm 1 combines the four algorithms in a way that optimally utilizes the common channel estimation values among the four algorithms (lines 7, 10, and 12 utilize the channel estimation values in $\mathbf{f} = (P_0, P_{01}, P_{02}, \cdots, P_{0n})$). $FastBeam$ first measures the fingerprint $\mathbf{f}$ of the current channel, and sees if there is a match (lines 1-3). If there is a match, it uses the corresponding beamformer $\mathbf{w}$ (lines 4 to 5). If there is no match in $FingerPrint$ (or the matched beamformer $\mathbf{w}$ is not satisfiable), an algorithm among the rest three will be selected according to the level of the variance V reported by the client. If V is "High," the base station performs the $ZeroPhases$ algorithm, which has the lowest time complexity

(line 12); if V is "Medium," the base station performs the $FewerAntennas$ algorithm (line 10); and if var is "Low," the base station performs the $FewerPhases$ algorithm. When $FewerPhases$ is carried out, it records the resulting beamformer $\mathbf{w}$ with the measured fingerprint $\mathbf{f}$ (lines 7 and 8). The database of $FingerPrint$ can be set up previously by carrying out dedicated training or purely learned during the operation. The timely learning of the fingerprint database makes $FastBeam$ adaptive to the environment.

3.6 Performance Evaluation

In this section, we use implementations of the $FastBeam$ and $SimpleBeam$ algorithms on an experimental multi-antenna AP platform called Phocus Array([14]) for performance comparison.

We implement the $FastBeam$ and $SimpleBeam$ algorithms on the Phocus Array ([14]) platform that has eight programmable antennas. The Phocus Array platform acts as an AP, and the server (controller) is a PC that directly connects to the AP. An off-the-shelf client wirelessly connects to the AP. We compare the throughput performance using iperf sessions between the client and the server under different situations. The CPU of the AP is XScale-Ixp42x with BogoMIPS equal to 532.48, and the CPU of the server is Intel Pentium 4 2.80GHz; the memory of the AP and the server are 128MB and 1GB, respectively. The experiment is carried out in a typical office environment. Since the possible movement of a client is composed of linear movement and rotation, we set up experiments when the client is moving in a straight line and when it is rotating. Also, since the change in the environment is typically caused by people, we set up experiments when people are crossing the LOS between the client and the AP periodically. Most of the indoor environments are combinations of these situations.

Fig. 4(a) shows the throughput ratio of $SimpleBeam$ and $FastBeam$ compared to Omni under different rates of client mobility. In this scenario, the client moves at a certain speed along a line trajectory back and forth. $FastBeam$ outperforms $SimpleBeam$ due to

Table 2: Memory and CPU usage of each algorithm

Algorithm	Memory(%)		CPU(%)	
	AP	Srv	AP	Srv
SimpleBeam	0.53	0.66	1.52	0.09
FastBeam	0.53	0.64	0.78	0.05
FingerPrint	0.53	0.61	0.49	0.03
ZeroPhase	0.53	0.61	0.26	0.02
FewerAntennas	0.53	0.63	0.48	0.03
FewerPhase	0.53	0.64	0.59	0.03

the decrease in time complexity. The improvement increases with the mobility speed (from 1.24x to 1.42x). Fig. 4(b) shows the throughput ratio of the two algorithms compared to Omni under different frequencies of LOS blocking by an interfering object. In this scenario, the client is static, and there is a human passing through the LOS between the client and the AP every few seconds. $FastBeam$ again outperforms $SimpleBeam$, and the improvements are from 1.27x to 1.78x. Note that $FastBeam$ performs better even when the passing frequency is 0. This is because $\mathbf{h}$ has small changes even under static environment, which could be observed in Fig. 3. Thus, $FastBeam$ could get benefit from its smaller time complexity even when the environment is static. Fig. 4(c) shows the throughput ratio of the two algorithms compared to Omni under different rates of client rotation. In this scenario, the client is put on a platform, and it rotates at a fixed speed. $FastBeam$ outperforms $SimpleBeam$, and the improvement increases with the rotation speed (from 1.27x to 1.67x). As shown in Fig. 4, $FastBeam$ outperforms $SimpleBeam$ under each situation due to the adaptive selection among the four algorithms that could lead to a good trade-off between time complexity and accuracy.

3.6.1 Micro Performance

We now present the throughput performance of the four algorithms, which together compose the $FastBeam$ algorithm. As can be seen in Fig. 5(a), $FewerPhases$ performs better than $SimpleBeam$ in every environment. $FingerPrint$ performs very well when there is a match, and the performance reduces to the same level as $FewerPhases$ when

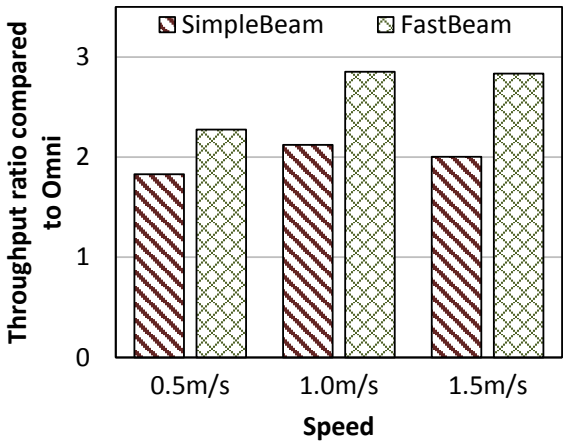
there is no match. Thus, overall $FingerPrint$ performs better than $FewerPhases$, and we can see that the fingerprint matching strategy does work.

The performance of $FewerAntennas$ is rather dynamic because it depends more on the characteristics of the amplitude of the channel gain $\mathbf{h}$ than the rate of client mobility. If the amplitude of the channel gains $\mathbf{h}$ is much larger on certain channels, $FewerAntennas$ can achieve close to the optimal received power with much less time complexity, which leads to better improvement. On the other hand, if the amplitude of the channel gains $\mathbf{h}$ is almost the same on each channel, $FewerAntennas$ will have less improvement. The $ZeroPhase$ performance is even more dynamic since its performance heavily depends on the phase difference θ_{ij} of the channel gain $\mathbf{h}$. If θ_{ij} is closer to 0 or π, $ZeroPhase$ will perform much better; on the other hand, if θ_{ij} is close to $\frac{\pi}{2}$, $ZeroPhase$ will perform sub-optimal. However, because of the reduction in time complexity, $ZeroPhase$ has better performance than $SimpleBeam$ when the moving speed becomes larger than 1.0m/s.

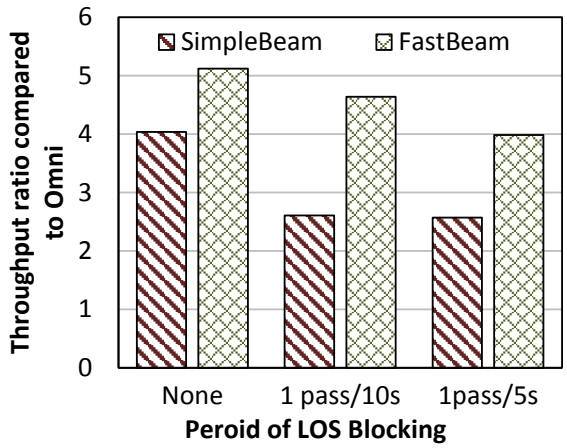
Figure 5(b) shows the time required to perform each algorithm. As can be seen, combining the four algorithms, $FastBeam$ is about 2x faster than $SimpleBeam$ on average and 4x when there is a match.

3.6.2 Overheads/Complexities

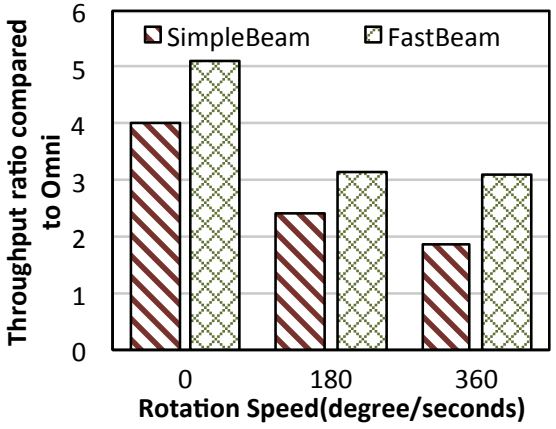
Table 2 shows the CPU and memory usage of each algorithm on AP and the server (Srv). As can be seen, the CPU and memory usage of all algorithms are both very small. The CPU usage is below 2%, and the memory usage is below 1% for the platforms used in the experiments.



(a) Client mobility

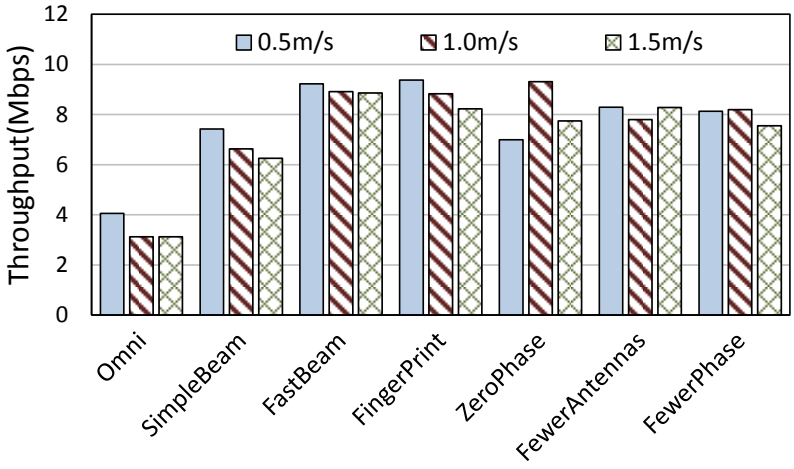


(b) LOS blocking

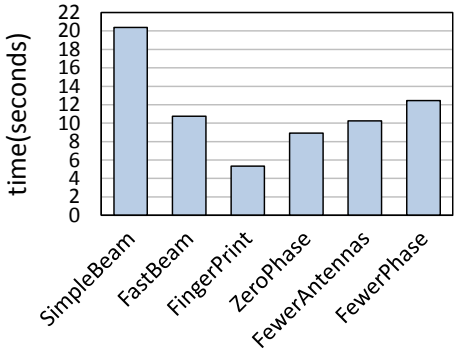


(c) Client rotation

Figure 4: Performance of $FastBeam$



(a) Throughput



(b) Convergence time

Figure 5: Performance of each algorithm

CHAPTER IV

SCHEDULED WIFI: ALGORITHMS FOR CONTROLLABILITY IN FUTURE-PROOFING NETWORKS

4.1 Introduction

Most WiFi deployments today use the Distributed Coordination Function (DCF) mode of the IEEE 802.11 standard [20]. The DCF mode of operation is simple and scalable and requires a participating node to listen to the channel and make contention decisions purely on locally available information. The contention algorithm, in turn, is controlled by a set of parameters including the maximum contention window that are adaptively adjusted based on local information. While the approach is simple and scalable, the goal of DCF is to achieve coarse-level fairness and efficiency in the network. Any finer-level goals are beyond the scope of DCF.

The IEEE 802.11 Point Coordination Function (PCF) mode, on the other hand, relies on centralized scheduling by the Access Point (AP) [20]. Theoretically, the scheduling algorithm at the AP can be arbitrarily defined. The problems with PCF are two-fold: i) it uses a polling process that incurs substantial overheads, especially under dynamic load conditions, and ii) the standard does not specify how APs should coordinate with each other to prevent collisions across cells.

There have been interests lately on the problem of achieving the benefits of centralized scheduling while retaining the simplicity and scalability benefits of distributed operations[1] [29]. The benefits of centralized scheduling are the following:

- Predictability: Applications and services that require predictable service can expect

[1] In a related domain, Software-Defined Networks (SDNs) aim to accomplish a similar goal (e.g., OpenFlow [12]).

to receive it in a setting with centralized scheduling. The central scheduler has complete control over what is transpiring in the network, and hence provide assurances.

- Differentiation: Applications and services can be provided with different resource allocations depending on their requirements. While there are distributed approaches to accomplish this goal (e.g., IEEE 802.11e [20, 73, 39]), they are quite coarse in the differentiation they provide.

- Efficiency: Finally, in environments where operational efficiency is an issue (e.g., in high-density WiFi deployments), centralized scheduling can lead to higher efficiencies.

On the other hand, the advantages of purely distributed operations are the lack of a single point of failure or bottleneck, scalability with the number of nodes, and backward compatibility with how WiFi is predominantly used today [8].

The context of this chapter is this bridge between centralized scheduling and purely distributed operations. We focus on how to *enable the controllability of WiFi networks without compromising their scalability*. We ask the following question: *Can the goals of centralized scheduling be achieved using purely distributed operations?* To achieve this aim, we introduce a MAC protocol called "*Rhythm*".

4.2 WiFi DCF - A Primer

The Distributed Coordination Function (DCF) mode of IEEE 802.11 ([20]) is a Carrier Sense Multiple Access (CSMA) MAC protocol. It belongs to the *listen before talk* family of protocols. Before a transmitter node (Tx) transmits, it senses the channel to determine if there are other nodes transmitting. If the channel is busy, the Tx [2] defers until the channel becomes idle. If the channel is idle for a specified duration (the DCF InterFrame Space (DIFS)), the Tx infers the channel to be idle and randomly selects a backoff number in [0,

[2] In this thesis, we use Tx to refer transmitter node and Rx to refer receiver node.

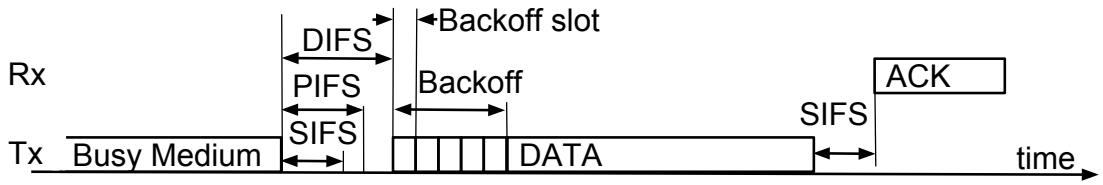


Figure 6: Timeline of DCF transmission

cw], where cw is the contention window. Then, the Tx counts down the backoff number in terms of backoff slots. If the channel becomes busy before the backoff timer expires, the Tx freezes its backoff and defers until the channel becomes idle again. Otherwise, the Tx transmits when the backoff number becomes zero. If the DATA transmission is successfully received, the receiver node (Rx) sends an ACK after a Short InterFrame Space (SIFS) duration. Fig. 6 shows the timeline for a DCF transmission. An optional mechanism, exchanging short control frames (RTS and CTS frames) before the data transmission, can be used to decrease the probability and impact of collisions but is rarely used due to its associated overheads.

4.3 Problem Definition

The goal of Rhythm is achieving scheduled WiFi with minimum overhead. Let us consider multiple infrastructure WLANs, containing n nodes, all using the same channel. In these WLANs, a central controller communicates with all APs. After gathering network-wide information (only macro-level information such as client lists from APs, not the queue status, is gathered), the central controller determines a target schedule $S = \{s_0, \cdots, s_{k-1}\}$, where $s_i \in N = \{0, \cdots, n-1\}$ indicates the scheduled node ID in position i. (Note that the length of schedule k can be larger than the number of nodes n if a node is scheduled multiple times.) S is delivered to all nodes through the APs. *With S, how can nodes in these networks efficiently contend distributedly (without any other communication with the central controller) to follow the transmission order in S?*

At a high-level, Rhythm involves changes to the contention mechanism in the 802.11

Table 3: List of Rhythm algorithms

Section	Algorithm	Considered Scenario
4.4.1	Rhythm-Base	Fully connected topology, nodes are usually backlogged
4.4.2	Rhythm-Shrink	Fully connected topology, having non-backlogged nodes
4.4.3	Rhythm-Clique	Partially connected topology, having non-backlogged nodes
4.4.4	Rhythm	Backward compatibility, other practical considerations

DCF. A Rhythm node listens to the ongoing transmissions and determines a *virtual schedule pointer* within S. It then contends based on the relative distance between the virtual pointer and its position within S. The non-trivial aspects of the Rhythm solution lie in how *work conservation* is achieved in the presence of non-backlogged nodes (Note that the queue status is unknown to the central controller) and how *partial connectivity* (i.e., nodes cannot overhear each other) are handled.

In the following section, we start with the simplest scenario and consider each problem gradually, as listed in Table 3.

4.4 Rhythm: Scheduled-WiFi using Distributed Contention

4.4.1 Baseline algorithm

Consider a scenario in which all nodes are in the transmission range of one another (i.e., a fully connected topology), and all nodes are usually backlogged. To make nodes follow the target schedule, we introduce a logic concept: the virtual schedule pointer.

4.4.1.1 Virtual schedule pointer

To follow a common schedule, nodes must be synchronized. Although the most straightforward way is time synchronization, fine-grained time synchronization can be difficult. Thus, Rhythm achieves scheduled WiFi by synchronizing nodes in a "logical schedule position," in which each node in the network maintains a virtual schedule pointer, Pos, that points to the current logical position in the schedule. If the values of Pos in all nodes are the same (i.e., they are synchronized), the nodes follow the target schedule by adjusting the backoff number. For example, let's assume that three nodes X, Y, and Z and a target schedule

$S = \{s_0 = X, s_1 = Y, s_2 = Z\}$ initially do a random backoff (the same way as DCF), and one node (assuming Y) wins the contention and transmits successfully. After hearing the successful transmission of Y, all nodes (including Y) update the virtual schedule pointer to 1. Then, instead of continuing the random backoff, Z sets its backoff number by calculating $D = pos(Z) - Pos = 2 - 1 = 1$, where $pos(Z)$ is the scheduled position of Z. Since $D = 1$, which means Z is the first node after Y, Z sets backoff number to 0, which makes it immidiately transmit after Y. Generalizing this concept, nodes use $((D - 1) \bmod k)$ as its backoff number, where k is the lenght of S. Similarly, X sets its backoff number to 1, and Y sets its backoff number to 2. If Z transmits, X resets its backoff number to 0 after overhearing the transmission of Z ($Pos = 2, pos(X) - Pos - 1 = 0 \bmod 3$). If Z does not transmit, X starts transmission when its backoff timer expires (X only waits for 1 backoff slot). Such a mechanism allows nodes to follow the target schedule without additional time synchronization. Employing this mechanism, we develop a baseline algorithm of Rhythm: Rhythm-Base.

4.4.1.2 Algorithm

We assume that target schedule $S = \{s_0, \cdots, s_{k-1}\}$ is known to all nodes in the network. All nodes maintain synchronization state ST and record $RC = \{r_0 \in N \bigcup \{Col\}\}$, where r_0 represents the most recent transmission, and Col represents a collision. Algorithm 2 illustrates position synchronization (UpdatePos) and schedule matching (MatchSch) of Rhythm-Base. Initially, every node does random backoff when RC is empty. Upon a successful transmission, the nodes update Pos to a position in S that matches RC. The update mechanism depends on the current synchronization state ST, which has two states: RAN and SYN. If $ST == RAN$, which indicates the lack of synchronization before this transmission, the nodes update Pos to *the smallest* matched position in S. If $ST == SYN$, nodes start from previous Pos and update Pos to the nearest matched position in S. After updating, each node sets the backoff number to $((D - 1) \bmod k)$, where

$D = pos_{nearest}(s_{self}) - Pos$ and $pos_{nearest}(s_{self})$ is the nearest scheduled position of each node. If a collision occurs, ST is reset to RAN, and the nodes perform a random backoff, just as they did in DCF. If a node hears a new transmission before the backoff timer expires, it updates Pos and resets the backoff number. Otherwise, when the backoff timer expires, the node transmits, records its transmission, and updates Pos.

Rhythm-Base is simple, proffers weighted fairness, as indicated in the target schedule, and yields near-optimal channel utilization: i) It wastes only one backoff slot (9μs in 802.11g) if a scheduled node does not transmit, ii) it causes no collisions when all nodes are synchronized, iii) it requires only one successful transmission for convergence in Pos. (Matching transmissions to the smallest position after collision avoids ambiguity when some nodes are scheduled multiple times.) Also, because of its fast convergence property, Rhythm-Base is robust against disturbances such as the packet error or loss of ACKs (both recorded as Col). Note that although Rhythm utilizes overheard transmissions, the active listening time is the same as that of DCF.

4.4.1.3 Overhead estimation

The use of Rhythm-Base introduces two overheads:

- *The overhead of broadcasting the target schedule:* Every node needs to know the target schedule S. S can be placed into a beacon every few seconds. The extra time for sending S is $T_s = \frac{k \times l}{R_b}$, where l is the length of the MAC address, k the schedule length, and R_b the sending rate of beacons. The overhead is $O_s = \frac{T_s}{T_p}$, where T_p is the period of updating the target schedule. If $R_b = 6Mbps$, $k = 100$, and $T_p = 100ms$, the overhead of broadcasting the target schedule is only $O_s = 0.8\%$, which is very low even when we have a long schedule and update it frequently.

- *The overhead of re-synchronization resulting from packet error and collision:* Assuming P_{col} is the probability of having a packet collision when n nodes contend using DCF, and P_{err} is the packet error rate, the overhead caused by packet errors

Algorithm 2 Rhythm-Base

function UPDATEPOS
  if r_0 ==Col || $|RC| == 0$ **then**
    set $RC = \{Col\}$
    $ST = RAN$
    return, and do regular random backoff as DCF
  else if $ST == RAN$ **then**
    Pos=MatchSch(-1)
  else
    Pos=MatchSch(Pos)
  end if
  set $RC = \{r_0\}$ ▷ clear old record
  $ST = SYN$
  set backoff_number=$(pos_{nearest}(s_{self}) - Pos - 1) \bmod k$
end function

function MATCHSCH($prevPos$)
  $j = (prevPos + 1) \bmod k$
  while True **do**
    if $s_j == r_0$ **then**
      return j
    end if
    $j = (j + 1) \bmod k$
  end while
end function

and collisions in DCF is $O_e = (1 - P_{col})P_{err} + P_{col}$. In Rhythm, 1 successful transmission can gain synchronization and avoid collisions, so the overhead in Rhythm is

$$O_e = P_{err} \times (1 - P_{col})(1 + P_{col} \times 2 + P_{col}^2 \times 3 + \cdots) \leq P_{err} \times \frac{1}{1 - P_{col}}.$$

If $P_{err} = 1\%$ and $P_{col} = 15\%$, then $O_e = 15.85\%$ in DCF and $O_e \leq 1.18\%$ in Rhythm.

4.4.1.4 Limitations of the baseline algorithm

Although Rhythm-Base yields near-optimal channel utilization with low overhead, it cannot perform well in the following situations:

- Non-backlogged nodes lead to inefficiency: In Rhythm-Base, one backoff slot (9μs) is wasted when a scheduled node does not transmit. When many nodes are continuously non-backlogged, the performance of Rhythm-Base might be worse than that

of DCF.

- Partial connectivity leads to a lack of information: Rhythm-Base achieves position synchronization by overhearing transmissions. Thus, if nodes are located outside the transmission range of the other nodes (i.e., hidden terminals), they do not overhear transmissions, and Rhythm-Base may not achieve position synchronization.

In the following subsections, we propose mechanisms that deal with these situations.

4.4.2 Work conservation with non-backlogged nodes

Non-backlogged nodes generate idle slots, leading to inefficiency. Instead of informing the central controller to change the target schedule, which causes extra overhead and experiences the delay between central entity and APs, we introduce a *Shrinking mechanism* in Rhythm (Rhythm-Shrink), in which nodes distributedly adjust the schedule.

4.4.2.1 Schedule shrinking for non-backlogged nodes

Each node maintains a *non-backlogged record*: If some scheduled nodes do not transmit, they are deemed *non-backlogged*. Non-backlogged nodes are ignored in backoff calculation and schedule-matching algorithms; that is, the schedule shrinks among these non-backlogged nodes. Schedule shrinking reduces idle slots and thus avoids the inefficiency caused by non-backlogged nodes. However, when nodes receive new packets, how do they get back on schedule? One simple way is to generate a collision that stops schedule shrinking. However, if 30 nodes are non-backlogged and only one of them receives new packets, a collision and 29 idle slots (261μs) is generated every time a node gets back on schedule. Thus, we introduce an efficient insertion mechanism that reinserts nodes on the schedule.

4.4.2.2 Mirrored collisions during insertions

If non-backlogged nodes exist, 1 backoff slot, referred to as an *insert slot*, is left at the end of each schedule cycle for these nodes to win the contention and start a transmission,

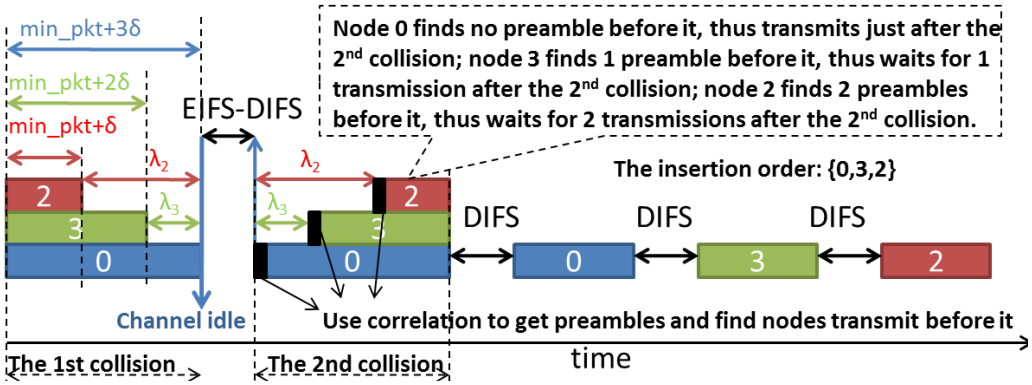


Figure 7: Timeline of mirror insertion

which stops schedule shrinking among the nodes. If only one node is inserted, it is recorded as backlogged when its transmission is heard. *However, how can several non-backlogged nodes be simultaneously inserted using only one backoff slot?* We introduce "*mirror insertion*," which generates two customized collisions for efficiently inserting multiple nodes simultaneously.

The main goal of mirror insertion is to determine an instant *insertion order* at which point all inserting nodes will be inserted so that no time is wasted waiting for non-backlogged nodes that are not inserting. Fig. 7 illustrates the timeline of mirror insertion. When nodes want to insert, they transmit a packet of a specific length (achieved by packet splitting and aggregating) in the base rate. The length is $min\_pkt + (n - I) \times \delta$, where min_pkt is a predefined minimum packet size known by all nodes, δ the sum of the preamble transmission time and the transmission switching delay and I the order of their first scheduled position in S. (For example, if $S = \{0, 3, 1, 3, 2\}$, the order of the first scheduled position is $\{0, 3, 1, 2\}$, and the insert packet length is min_pkt for node 2, and $min\_pkt + 3 \times \delta$ for node 0.) When multiple nodes are inserted simultaneously, a collision occurs. Each inserting node records λ_i ($i \in N$), the time from the end of its transmission to the time when the channel becomes idle. Then, each inserting node transmits at $EIFS - DIFS + \lambda_i$ after the channel becomes idle, creating a second collision, a "mirror image" of the first. After the first collision, all inserting nodes listen to the channel before transmitting. Since the

start of each transmission is separated by at least δ, each node can hear the preambles of all prior transmissions before starting its own. Each node then applies preamble correlations to the received signal and figures out how many nodes start transmitting before it starts to transmit during the second collision. This transmission order determines the insertion order for all inserting nodes, as illustrated in Fig. 7.

The key mechanism of mirror insertion is *identifying preambles in a collision using correlation*. A preamble is a pseudo-random sequence that can be identified using correlation, even under high interference. Studies have proven this mechanism valid with multiple random collided packets [36]. Since mirror collision is designed in a way in which preambles are separated in the integer number of δ (we use 26μs when the preamble transmission time is 16μs), its identification is even easier.

Mirror insertion does not require fine-grained time synchronization. As long as the preambles in the second collision are separated far enough for identification, an insertion order is determined. Since mirror collisions are two collisions within $EIFS$, they are easily recognized and are not treated as regular collisions.

4.4.2.3 Algorithm

Algorithm 3, 4, and 5 illustrate the position synchronization, schedule matching, and backoff calculation of Rhythm with shrinking mechanism.

In Algorithm 3, mirror collision is recorded as *M_Coll* in RC, and is treated differently from normal collision (line 2). The non-backlogged record is cleared when the transmission of a node is heard (line 13) or when ST becomes RAN (line 5).

When determining Pos in Algorithm 4, insertion is recognized by transmissions from non-backlogged nodes or mirror collisions (line 2), and Pos is set to k during the insertion process (line 6). Nodes are recorded as non-backlogged if the location of Pos "jump" among them, which means that they didn't transmit (line 3 to 5 and line 13 to 15).

Algorithm 5 illustrates the backoff calculation of Rhythm-Shrink. When a non-inserting

node calculates the backoff number, it skips non-backlogged nodes (line 13) and adds an insert slot if required (line 14 to 15). Inserting nodes schedule their transmission to the end of schedule cycle (line 5 to 6). After creating mirror collisions, inserting nodes learn the insertion order, continue counting the insertion transmissions, and set up the backoff number accordingly. In line 8, (n_front_insert) is the number of preambles the node found before it in the second collision of the mirror collisions. n_insert is the number of insertion transmissions that already happened. Since $Pos = k$ during insertion, non-inserting nodes will continue adding an extra backoff slot (line 14 to 15), and inserting nodes win the contention after creating mirror collisions.

Algorithm 3 Rhythm-Shrink

```
1:  function UPDATEPOS
2:      if r_0 ==Col || |RC| == 0 then        ▷ mirror collision M_Coll will not be treated as Col
3:          set RC = {Col}
4:          ST = RAN
5:          clear the NonBacklogged record of all nodes
6:          return, and do regular random backoff as DCF
7:      else if ST == RAN then
8:          Pos=MatchSCH(-1)
9:      else
10:         Pos=MatchSCH(Pos)
11:     end if
12:     set RC = {r_0}
13:     clear the NonBacklogged record of node r_0
14:     ST = SYN
15:     set backoff_number=CalBK(Pos)
16: end function
```

4.4.2.4 Overhead estimation

Mirror collisions result in the main overhead of the shrinking mechanism:

$$O_{mirr} = \frac{2 \times T_{col} \times P_{insrtCol}}{\lceil \frac{T_{inact}}{T_{sch}} \rceil \times T_{sch}}$$

, where $T_{col} = T_{min\_pkt} + (n - I_{avg}) \times \delta$ is the time spent in each mirror collision, I_{avg} is the average minimum first scheduled position of the insert nodes, $P_{insrtCol}$ is the probability of having more than two inserting nodes, T_{inact} is the average period in which a node runs

Algorithm 4 Match function of Rhythm-Shrink

function MATCHSCH($prevPos$)
  if $r_0 \in$ NonBacklogged $\parallel$ $r_0 ==$*M_Coll* **then**
    for $i = prevPos + 1$ to $k - 1$ **do**
      set s_i NonBacklogged
    end for
    return k
  end if
  $j = (prevPos + 1) \bmod k$
  while True **do**
    if $s_j == r_0$ **then**
      return j
    end if
    if $ST == SYN$ **then**
      set s_j NonBacklogged
    end if
    $j = (j + 1) \bmod k$
  end while
end function

Algorithm 5 Backoff Calculation of Rhythm-Shrink

function CALBK(Pos)
  if $IsInsterting$==False **then** ▷ node is not inserting
    Sch_Pos=$pos_{nearest}(s_{self})$
  else if $IsInsterting$==True **then** ▷ node is inserting
    if $InstST$==Before_Mirror_Coll **then** ▷ before mirror collisions
      Sch_Pos=k
    else if $InstST$==After_Mirror_Coll **then** ▷ after mirror collisions
      BK =(n_front_insert)-(n_insert)
      return BK
    end if
  end if
  $BK = (\text{Sch\_Pos} - Pos - 1) \bmod k$;
  $BK = BK -$NonBkLogCount(Pos,Sch_Pos) ▷ skip non-backlogged nodes
  if Sch_Pos$\leq Pos$ & NonBkLogCount(0,k)> 0 **then**
    BK++
  end if
  return BK
end function
function NONBKLOGCOUNT($p1$,$p2$)
  return the number of non-backlogged nodes between position $p1$ and $p2$
end function

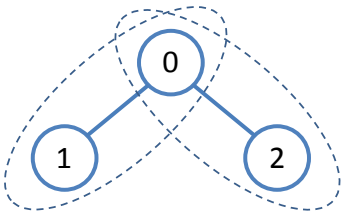


Figure 8: Example of three nodes

out of packets, T_{sch} is the average period of the target schedule, and T_{min_pkt} is the transmit time of a packet with size min_pkt.

T_{col} slightly increases as the number of node increases: If $T_{min\_pkt} = 52\mu s$, $\delta = 26\mu s$, $n = 50$, $I_{avg} = 25$, $P_{insrtCol} = 1$ and $\lceil \frac{T_{inact}}{T_{sch}} \rceil \times T_{sch} = 50ms$, we have $O_{mirr} = 2.81\%$. If $n = 300$, $I_{avg} = 150$, we have $O_{mirr} = 15.81\%$. Compared to 40%, which results from collisions when DCF is used with $n = 300$, this value is still reasonable. The traffic dynamic and the schedule period also affect O_{mirr}. If $T_{sch} = 200ms$ or $T_{inact} = 200ms$, $O_{mirr} = 3.95\%$ in the $n = 300$ case.

In summary, the overhead grows only when the three following conditions are satisfied: i) the number of nodes is large, and ii) traffic is strongly dynamic, and iii) the average schedule period is short. Because it is unlikely to have both short schedule period and a significant number of nodes, Rhythm can yield near-optimal channel utilization in most practical situations with various traffic loads.

4.4.3 Handling partial connectivity

Partial connectivity hinders position synchronization in Rhythm. Instead of transmitting RTS/CTS, which generates significant overhead, we introduce a *Clique mechanism* in Rhythm (Rhythm-Clique).

4.4.3.1 Separation and connection

The main concept of the Clique mechanism is *separating nodes that cannot overhear each other into different groups*. Nodes inside the same group overhear each other. Thus, each group can operate Rhythm-Shrink by itself. Then, we use *nodes that can hear different groups for connecting groups.* We illustrate this concept using an example in Fig.4.4.3, where nodes 1 and 2 cannot hear each other. The target schedule separates the two nodes and connect them using node 0: $S = \{0, 2, 2, 0, 1, 1\}$. The start and end positions of each group are the positions of the adjoining connecting nodes (indicated in Fig. 9(a)). When Pos points to positions inside a group (referred to as *active positions*), nodes in the group operate Rhythm-Shrink. Otherwise, they set up a long backoff. After the first successful transmission of node 0, all nodes have $Pos = 0$. Node 1 sets up a long backoff, and node 2 transmits twice. Then, node 0 transmits again, which updates $Pos = 3$. Node 2 sets up a long backoff, and node 1 transmits twice. Then, node 0 transmits and updates $Pos = 0$ again. Repeatedly, the nodes follow the schedule even when they cannot overhear each other. Based on the same concept, we describe the Clique mechanism below.

4.4.3.2 Cliques and Bridges

The Clique mechanism involves a target schedule that separates nodes that cannot overhear each other into different portions, or Cliques, of the schedule (all nodes inside a Clique overhear each other) and places "Bridges" to connect the Cliques. The start and end positions of a Clique are the positions of the adjoining Bridges, which define the *active positions* of a Clique. Fig. 9(b) shows a schedule with three Cliques, C_i, four Bridges, B_j, and start/end positions.

To trigger Pos updates, Bridges should always transmit, and the transmissions of Bridges must be heard by their adjoining Cliques. Accordingly, Bridges can be set up in three ways:

- *Using a node that hears both Cliques:* A node that hears both Cliques can act as a Bridge connecting them.

- *Using nodes from both Cliques that hear each other:* shown in Fig. 9(b); after overhearing the transmission of B_1, the Pos update of $C1$ is triggered, and B_2 transmits and triggers the Pos update in $C2$.

- *Using nodes from both Cliques that can communicate with each other using the backbone:* Some nodes can communicate using a backbone connection (such as APs). In Fig. 9(b), when B_{2b} transmits, it sends a packet containing the transmission time of its wireless transmission to B_{3b} through the backbone. Based on information carried in this packet, B_{3b} learns the end time of transmission of B_{2b}, and transmits to trigger the Pos update in $C3$.

We can extend the concept of the Clique mechanism to scheduling multiple Cliques that do not interfere with each other in parallel (Fig. 9(c)).

4.4.3.3 Algorithm

Assume that the conflict graph of WLANs is known by the central controller (The conflict graph of WLANs can be generated by an online passive interference estimation, PIE [67]. We will give more illustrations in Section 4.4.4). The controller designs the target schedule S with Cliques and Bridges (each client belongs to at least one Clique formed by its AP and itself). Since Bridges always need to transmit, only APs are selected as Bridges. If a Bridge runs out of data packets, it transmits a CTS-to-Self with NAV$= 0$. The first position s_0 is always a Bridge. After separating nodes into Cliques, the schedule is re-weighted depending on the requirement. S with information about the Cliques and Bridges is then delivered to all nodes.

Algorithm 6, 7, and 8 illustrate the position synchronization, schedule matching, and backoff calculation of Rhythm with Clique mechanism. In Algorithm 6, all nodes (except s_0) set a long backoff time initially (line 2 to 8). s_0 starts the first transmission and nodes update Pos when they overhear transmissions. If Pos belongs to its active positions, a node operates as Rhythm-Shrink (line 22 to 23). Otherwise, it sets a long backoff time and waits

for Pos being triggered to its active positions (line 25). In Algorithm 7, each Clique does insertion independently. That is, nodes maintain a non-backlogged record only for nodes in the Clique they belong. Besides, an insert slot is added at the end of each Clique (line 6). In Algorithm 8, inserting nodes set up its scheduled position to the end of its Clique (line 6).

Algorithm 6 Rhythm-Clique

```
 1: function UPDATEPOS
 2:     if |RC| == 0 then
 3:         if s_0 == s_self then
 4:             start to transmit
 5:         else
 6:             set up long time backoff
 7:         end if
 8:     end if
 9:     if r_0 ==Col || |RC| == 0 then
10:         set RC = {Col}
11:         ST = RAN
12:         clear the NonBacklogged record of all nodes
13:         return, and do regular random backoff as DCF
14:     else if ST == RAN then
15:         Pos=MatchSCH(-1)
16:     else
17:         Pos=MatchSCH(Pos)
18:     end if
19:     set RC = {r_0}
20:     clear the NonBacklogged record of node r_0
21:     ST = SYN
22:     if Pos ∈ ActP then
23:         set backoff_number=CalBK(Pos);
24:     else
25:         set up long time backoff
26:     end if
27: end function
```

4.4.3.4 Overhead estimation

Rhythm-Clique contains three major overheads:

- *Broadcasting extra information and dynamically generating conflict graphs:* Similar to broadcasting S, broadcasting additional information about Cliques and Bridges result in little overhead. For generating conflict graphs dynamically, we can utilize PIE [67], which is an online passive interference estimation for WLANs. PIE, which has

Algorithm 7 Match function of Rhythm-Clique

function MATCHSCH($prevPos$)
  if $r_0 \in$ NonBacklogged $\parallel$ $r_0 ==$*M_Coll* **then**
    for $i = prevPos + 1$ to $k - 1$ **do**
      set s_i NonBacklogged
    end for
    return EndPosClique
  end if
  $j = (prevPos + 1) \bmod k$
  while True **do**
    if $s_j == r_0$ **then**
      return j
    end if
    if $ST == SYN$ **then**
      set s_j NonBacklogged
    end if
    $j = (j + 1) \bmod k$
  end while
end function

Algorithm 8 Backoff Calculation of Rhythm-Clique

function CALBK(Pos)
  if $IsInsterting$==False **then** ▷ node is not inserting
    Sch_Pos=$pos_{nearest}(s_{self})$
  else if $IsInsterting$==True **then** ▷ node is inserting
    if $InstST$==Before_Mirror_Coll **then** ▷ before mirror collisions
      Sch_Pos= EndPosClique
    else if $InstST$==After_Mirror_Coll **then** ▷ after mirror collisions
      BK =(n_front_insert)-(n_insert)
      return BK
    end if
  end if
  $BK = (\text{Sch\_Pos} - Pos - 1) \bmod k$;
  $BK = BK -$NonBkLogCount(Pos,Sch_Pos) ▷ skip non-backlogged nodes
  if Sch_Pos$\leq Pos$ & NonBkLogCount(0,k)> 0 **then**
    BK++
  end if
  return BK
end function
function NONBKLOGCOUNT($p1$,$p2$)
  return the number of non-backlogged nodes between position $p1$ and $p2$
end function

very low overhead, fast convergence time, and fine-grained interference estimation, can handle mobile clients at walking speed (0.25 m/s).

- *Transmitting CTS-to-self packets:* The overhead of transmitting CTS-to-self packets is $O_B = \frac{P_{AP} \times n_B \times CTS}{T_{sch}}$, where n_B is the number of Bridges, P_{AP} the probability that a Bridge (which is also an AP) runs out of packets, CTS the transmission time of a CTS packet, and T_{sch} the average period of the schedule cycle. Typically, because the downlink traffic is heavy [66], P_{AP} is small; since $k \geq n >> n_B$, $T_{sch} >> n_B \times CTS$. Thus, O_B is small.

- *Backbone transmission delay between Bridges:* The overhead introduced by delay in the backbone is $O_d = \frac{d \times n_{Bd}}{T_{sch}}$, where d is the average delay deviation between two Bridges and n_{Bd} the number of Bridges that communicate through the backbone. Consider $k = 30$ and $n_{Bd} = 2$, $O_d = 9.17\%$, even when $d = 0.5ms$. (The average delay d between two APs with one switch is 0.1ms with a standard deviation of 0.09ms [23].)

4.4.4 Other challenges and considerations

4.4.4.1 System architecture

Fig. 10 shows the system architecture of WiFi networks operating Rhythm. The system architecture of WiFi networks operating Rhythm contains a central controller that periodically communicates with these networks. The central controller can be a WiFi controller or a server set up in the cloud. WiFi network information, such as the conflict graph and the client list, are periodically sent to the central controller. The conflict graph can be obtained by using PIE [67], and the client list can be monitored by each AP. Protocols such as CAPWAP [18] and LWAPP [19] allow APs to communicate with a central controller. Based on the collected information, the central controller determines the target schedule S with Cliques and Bridges and delivers it to all nodes through APs.

4.4.4.2 Schedule updates and membership changes

The update of S is indicated by a sequence number carried in ACK sent by AP (ACK contains four unused bits [20]). If nodes receive or overhear a new sequence number, they switch to using non-Rhythm duration until they receive or overhear the new S from AP. When a node wishes to join a network, it first tries to overhear S from the APs, and then uses the non-Rhythm period for registration. Note that since Rhythm automatically shrinks the schedule, a delay in updating the target schedule does not decrease channel utilization.

4.4.4.3 Scheduling decision

Currently, the central controller simply uses greedy algorithms to locate Cliques in the conflict graph and places all nodes on the target schedule. However, since Rhythm controls relative scheduling and weighted fairness, with more intelligent scheduling decisions, it is possible for Rhythm to solve other MAC issues, such as QoS, energy-efficient, and higher layer traffic aware scheduling. In future work, we will explore the potential application of Rhythm in other MAC issues.

4.5 Evaluation

In this section, we present experimental results carried out by WARP and use ns2 simulations to evaluate each mechanism in Rhythm.[3] Then, we compare Rhythm to a closed related work, Domino [78]. Table 5 shows the simulation parameters, which follows 802.11g.

4.5.1 WARP experiments

We implement Rhythm in a software-defined radio platform: the Wireless Open-Access Research Platform (WARP) v3 [15]. We set up a fully connected topology in a typical indoor environment with three WARP nodes (Figure 11), one that acts as an AP, and the other two that act as clients, operating in 5.18GHz with 54Mbps data transmission rate. Iperf is

[3] Since the WiFi backoff timer duration is "lazily calculated" in ns-3 [11], and the backoff timer is vital for Rhythm, we evaluate Rhythm using ns-2.

Table 4: Throughput comparison of Rhythm and DCF in experiments

Link Number	Rhythm (Mbps)	DCF (Mbps)
1	9.00	6.62
2	8.18	8.98
3	9.00	6.62
4	8.18	9.00
Total	34.36	31.22

Table 5: ns-2 parameters

Parameter	Value
Frame size	1500byte
Basic transmission rate	6Mbps
Data transmission rate	54Mbps
Slot time	9μs
SIFS	10μs
DIFS	28μs
δ	26μs

used to generate UDP traffic with frame size 1500byte. Table 4 shows the throughput of each link (two uplinks and two downlinks) from the experimental evaluation. Rhythm produces a throughput which is 95% of the theoretical optimal (36Mbps) and better fairness than DCF (the target schedule S is: $\{$AP, client1, AP, client2$\}$). The two uplinks (link 2 and 4) have slightly better throughput in DCF. It is because DCF roughly gives each node the same access rate, which gives each node approximately $\frac{1}{3}$ access rate. Rhythm gives each link the same access rate, which gives each link approximately $\frac{1}{4}$ access rate. Since $\frac{1}{4} < \frac{1}{3}$, links 2 and 4 get slightly higher throughput in DCF. Given the limited number of hardware, the throughput difference between Rhythm and DCF is not large. In scenarios with more nodes in the following sections, large improvement will occur. We consider these experimental results as a proof of concept and evaluate the performance of Rhythm in more complicated situations using ns-2 simulations.

4.5.2 Baseline algorithm

Fig. 12 shows the time usage (channel utilization) of Rhythm in a topology with 20 nodes (all nodes are backlogged, and the topology is fully connected). As expected, Rhythm spends almost no time in collision or backoff idle, and the small portion of idle time results

from Interframe Space (IFS), which is an unavoidable protocol overhead, yielding channel utilization of almost 90%. Compared to DCF, Rhythm leads to 20% improvement in channel utilization. We also examine the ability of Rhythm to provide weighted fairness. Using the same topology, we randomly assign schedule weights to each node and calculate the weighted Jain's fairness index. Fig. 13 shows that Rhythm produces correct weighted fairness as indicated in the target schedule.

4.5.3 Shrinking mechanism

We examine the shrinking mechanism of Rhythm using fully-connected topologies and a variety of traffic. First, we present the performance of Rhythm-Base at determining the effect of the shrinking mechanism in a topology with 50 nodes. The two APs are always backlogged, and the clients are randomly backlogged periodically during a certain percentage of the time. In Fig. 14, while the channel utilization of Rhythm-Base decreases as the average backlogged time decreases, Rhythm maintains channel utilization.

4.5.4 Clique mechanism

To examine the performance of the Clique mechanism of Rhythm, we design various topologies and traffic loads. Since the overhead of the mechanism is not obvious when the number of nodes is large, we decrease the number of nodes to 30. Two APs are located in the center act as Bridges; we change the locations of the clients to create hidden terminals. All nodes are continuously backlogged. Figs. 15(a) and 15(b) show channel utilization and fairness under varied percentages of nodes suffering from hidden terminal problems. With the Clique mechanism, Rhythm yields near-optimal channel utilization and fairness, as indicated in the target schedule. Next, to examine the overhead of transmitting CTS-to-self packets, we change the backlogged time percentage of the two APs. As shown in Fig. 16(a), since $n = 30 >> n_{Barr} = 2$, the overhead is small even when the backlogged percentage of APs is 0%. Finally, to estimate the overhead caused by backbone delays, we separate the two APs and ensure that they cannot hear each other and communicate

through the backbone. As shown in Fig. 16(b), channel utilization decreases by 9% when the average backbone delay reaches 0.5ms, which agrees with our overhead estimation.

4.5.5 Comparison with Domino

We compare the performance of Rhythm with closely-related Domino [78]. Since the WiFi backoff timer duration is "lazily calculated" in ns-3 [11], we evaluate Rhythm using ns-2. However, as the simulation of Domino is available in only ns-3, we implement Rhythm-Base (representing Rhythm) in ns-3 and compare it to Domino in a fully connected topology (no hidden terminal and no exposed terminal) with all nodes continuously backlogged.

Similar to Rhythm, Domino [78] triggers the next transmission by overheard transmissions. (The batch design in Domino is similar to the Clique mechanism in Rhythm.) While the target schedule in Rhythm is a high-level transmission order in which nodes have a certain degree of distributed adjustment (e.g., the shrinking mechanism), the schedule in Domino is an exact transmission schedule that nodes always follow. Thus, Domino needs to collect the queue status from all nodes and generates a schedule based on the queue status. Since the collection of queue status generates certain overheads, the throughput of Domino is lower than Rhythm, as shown in Fig. 17. The channel utilization of Domino is 76% while that of Rhythm can reach 88%. Also, since Domino generates a schedule based on the queue status, the schedule can be biased to flows with a larger queue size, especially when the batch size is large (i.e., it schedules all packets for flows with a smaller queue size and thus extra slots for flows with a larger queue size). As indicated in Fig. 18, when certain flows have a smaller queue size, Domino does not ensure as much fairness as Rhythm.

4.6 Case-Studies for Applying Rhythm

In this section, we identify 4 scenarios where the implementation of Rhythm provides significant performance improvement.

4.6.1 Efficient high-density WiFi deployments

As the number of WiFi nodes increases, the overall throughput decreases due to collisions. Wireless node manufacturers like Cisco-Meraki [28] and Ruckus [58] have new designs for APs to handle high density WiFi networks. They utilize beamforming techniques, centralized channel allocation, load balancing and interference-mitigation techniques to improve channel utilization. These techniques require new hardware and careful deployment of APs.

Rhythm can help improve the channel utilization in high density WiFi without the requirement of new hardware and the deployment effort. We carry out ns-2 simulations to evaluate the performance of Rhythm. The central controller fairly schedules each node in the network. Fig. 19 shows the channel utilization of Rhythm, DCF, and DCF with RTS/CTS in a topology where different number of nodes randomly deployed in a 500m$\times$500m area. Two APs locates in the center portion and can be heard by all nodes. All the clients has random traffic, while the two APs are always backlogged. As shown in Fig. 19, since Rhythm greatly reduces collisions and contentions by following the target schedule, it achieves good channel utilization even when the number of nodes is as large as 300. This results also evaluate the overhead of mirror insertions (Equation (7)). Initially, the channel utilization of Rhythm slightly decreases since T_{col} slightly increases as the number of nodes increases, which increases the overhead of mirror insertions. However, since T_{sch} and T_{inact} also increase as the number of nodes increases, this decrease slows down. Finally, Rhythm achieves over 200% improvement in channel utilization when the number of nodes reaches 300.

4.6.2 Power-saving with precise sleep patterns

In order to save power, WiFi nodes can be put to sleep, and many protocols have been proposed to make nodes efficiently sleep and listen when idle ([31, 41, 40, 77]). Although WiFi nodes can go to sleep when there is no packet to transmit, nodes need to keep listening

Table 6: Average power consumption of Nokia N810

Operating mode	Average Power(mW)
Idle	884
Receive	1181
Transmit	1258
Sleep	42

to the channel for contention when there are packets in the queue. Table 6 shows the average power consumption of a WiFi interface ([72]). Since receiving (i.e., listening) consume rather large power, power consumption when nodes are contending is huge.

Rhythm can decrease the power consumption of contending. Since nodes transmit following a target schedule, the time before its next transmission can be estimated. Thus, nodes can go to sleep even when there are packets in the queue. If the estimation is too large, and nodes find themselves being passed after sleeping, it can insert in the end of its clique. We evaluate the power saving potential of Rhythm using ns-2 simulations under a simple scenario where nodes always transmit. The power consumption parameters are in Table 6, and the transition of operation mode (e.g., receive to sleep or sleep to receive) wastes 50mJ each time. Since nodes always transmit, it is easy to estimate the next transmit time after transmission. Thus, nodes go to sleep after transmission, and wake up 1 transmission time prior to its transmission for updating Pos, and transmit again. Fig. 20(a) shows the power consumption of each operation per node when operating Rhythm, DCF, and DCF with RTS/CTS. Although Rhythm spends more power on transition from receive to sleep and from sleep to receive, the saved power on receive is tremendous. Fig. 20(b) shows the total power consumption per node of each protocol. It shows that more than 50% of power is saved when using Rhythm. This indicates a promising direction of further power saving for WiFi nodes when operating Rhythm.

4.6.3 Quality of service with weighted fairness

Weighted fairness for quality of service is an important requirement for many applications. Since DCF can not provide weighted faireness, mechanisms that focus on supporting weighted fairness in WiFi networks has been proposed ([56, 24]).

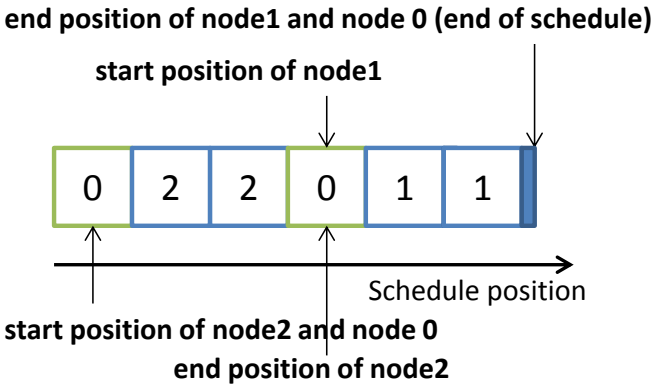
The schedule driven property of Rhythm makes it easy to achieve weighted fairness in WiFi networks. The central controller designs the target schedule based on the required weight, and nodes get the required weight simply by operating Rhythm. We use ns-2 simulation to evaluate the ability of Rhythm to provide weighted fairness. In a topology with 20 nodes containing 2 APs, we randomly assign schedule weights to each node and calculate the weighted Jain's fairness index (i.e., the throughput of each node is divided by its weight, and then used for calculating Jain's fairness index). Fig. 21(a) shows that Rhythm produces correct weighted fairness. We further set up a specific target schedule to evaluate the short term fairness. The target schedule has 4 nodes with weight 4, 8 nodes with weight 2, and 8 nodes with weight 1. Fig. 21(b) shows the average throughput of each node, and the required weight is being followed in average. Fig. 21(c) further shows the average time between each transmissions for each node, which indicates the short term weighted fairness. The average time is also inversely propotional to the weight. This shows that it is possible to give a fine-grained weighted fairness by operating Rhythm.

4.6.4 TCP-aware channel allocation

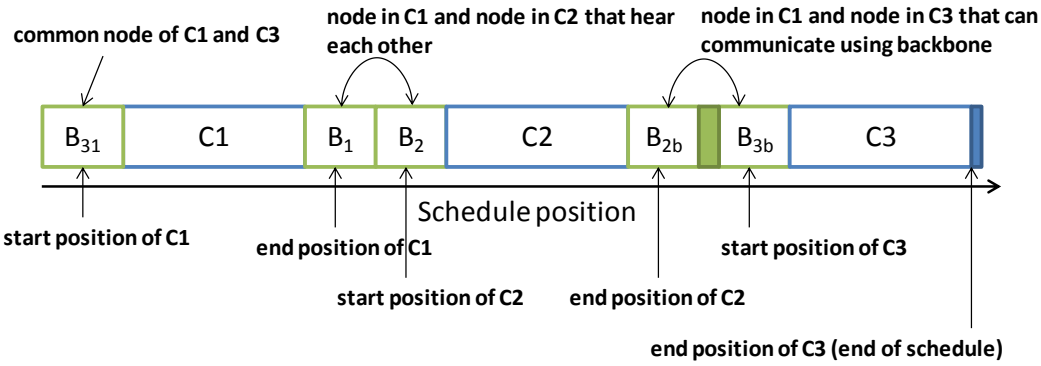
The sending rate of TCP relies on the feedback of TCP ACK. Thus, improper MAC decision can lead to unfairness between TCP flows. In WiFi networks, DCF roughly gives equal transmission probability to each node (not flow). Thus, TCP flows can suffer severe unfairness in certain situations. For example, consider a network with one AP and 10 nodes, each has a TCP uplink flow. While all nodes have the same access rate, AP needs to send TCP ACK to 10 nodes. Since AP cannot send back ACKs to all nodes efficiently, nodes receiving ACK earlier will generate higher TCP sending rate than the node receiving ACK later, which leads to unfairness. Thus, protocols that try to optimize TCP performance from MAC layer have been proposed in different networks ([54, 34]).

Rhythm can solve this unfairness problem by adjusting the target schedule. Given the current number of TCP flows in the network, the central controller schedule nodes based

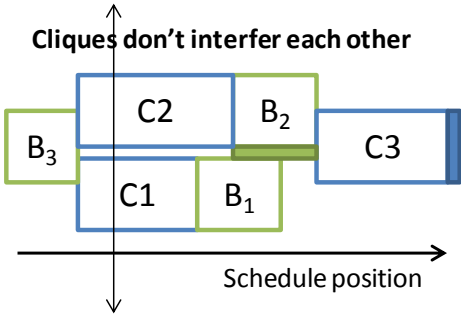
on the number of flows they contain. We use ns-2 simulations to evaluate the ability of Rhythm to provide fairness for TCP flows. We set up a topology with two APs and different number of TCP uplink flows. Fig. 22(a) and 22(b) show the aggregate throughput and Jain's fairness index under different number of TCP flows. Since TCP ACK can come back to the clients in time, Rhythm achieves great fairness with higher aggregate throughput than DCF and DCF with RTS/CTS. Fig. 22(c) shows a detailed look at a scenario with 26 flows. The throughput across flows varies from 0 to 2.5Mbps under DCF, while flows have the same throughput under Rhythm.



(a) Schedule of three nodes



(b) Different Bridges



(c) Parallel Cliques

Figure 9: Schedules and topologies of Clique mechanism

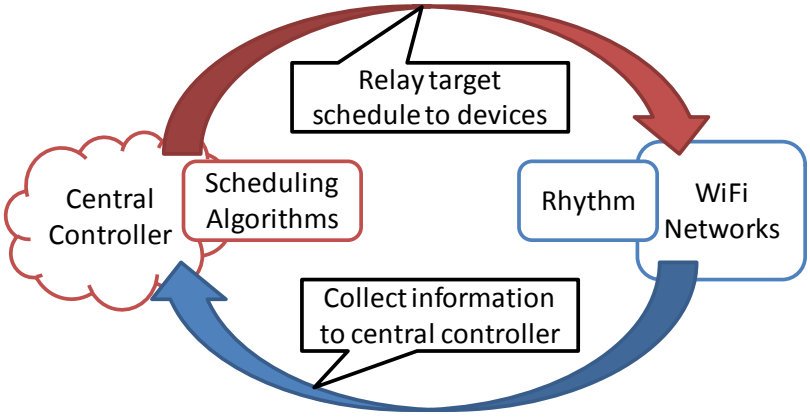


Figure 10: System architecture of Rhythm

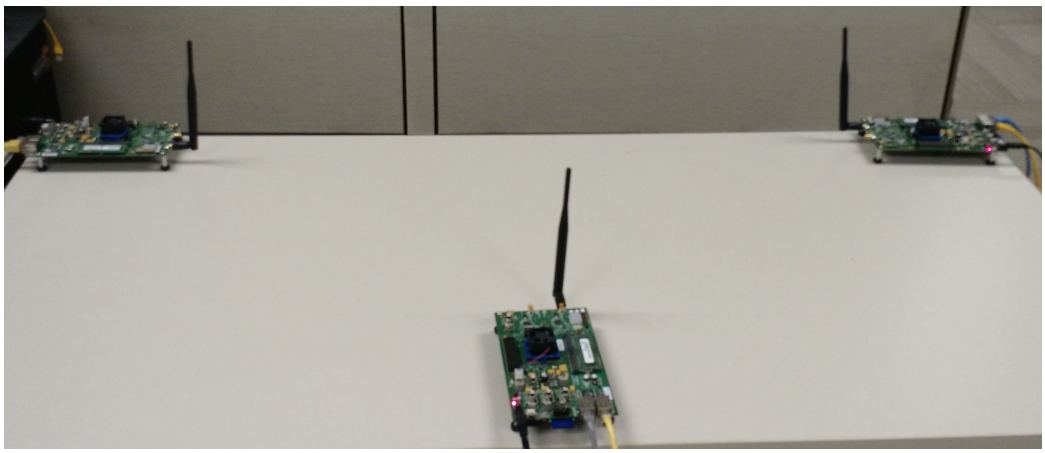


Figure 11: WARP test-bed

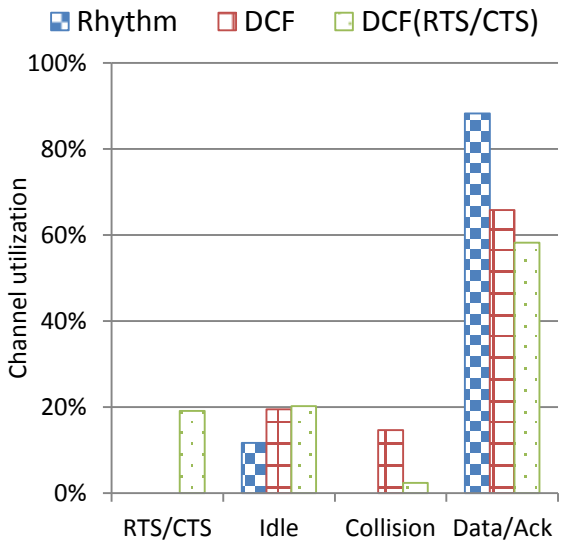


Figure 12: Time usage of Rhythm

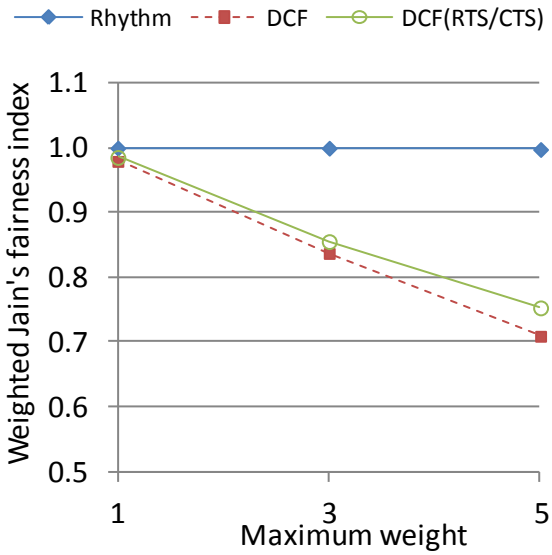


Figure 13: Weighted fairness of Rhythm

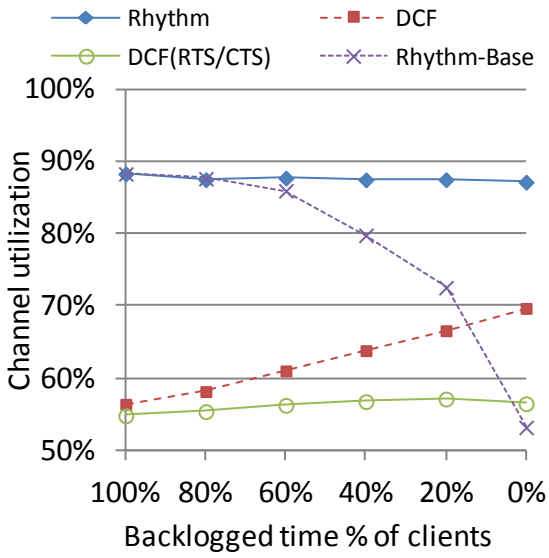
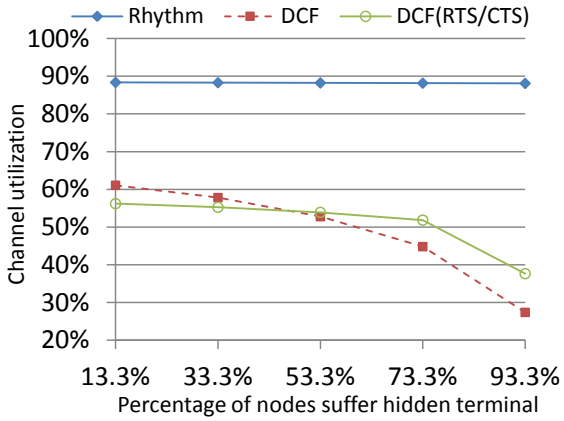
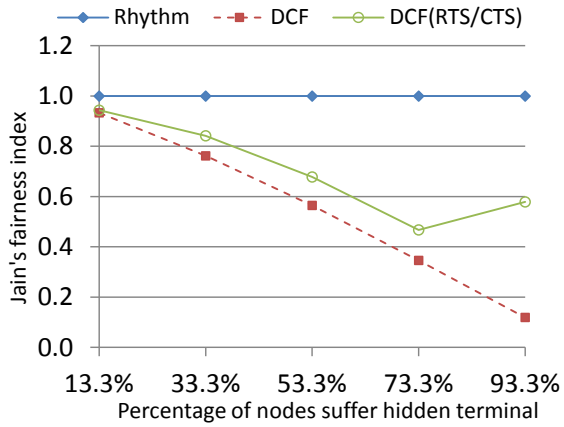


Figure 14: Evaluation of shrinking mechanism

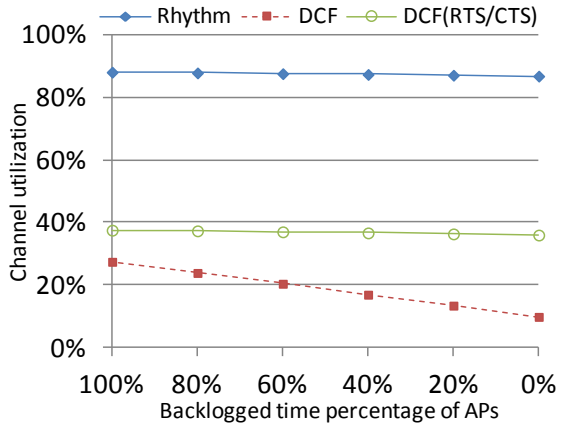


(a) Channel utilization with hidden terminals

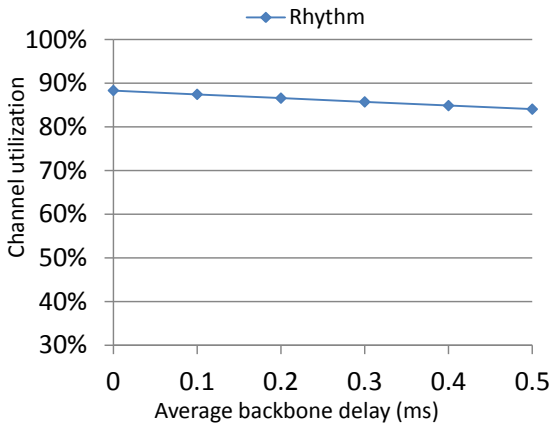


(b) Fairness with hidden terminals

Figure 15: Evaluation of Clique mechanism (1/2)



(a) Channel utilization with non-backlogged Bridges



(b) Channel utilization with backbone transmission delay

Figure 16: Evaluation of Clique mechanism (2/2)

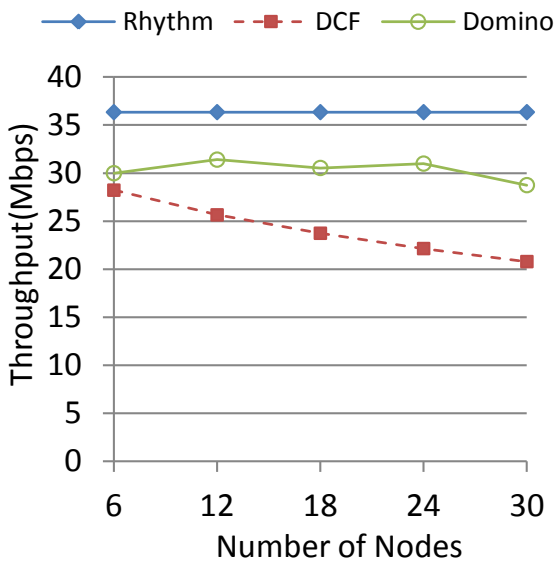


Figure 17: Throughput comparison in fully connected topologies

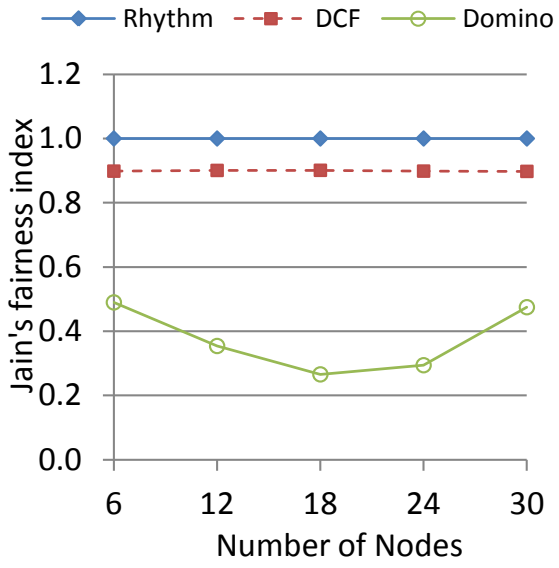


Figure 18: Fairness comparison when some flows have a smaller queue size

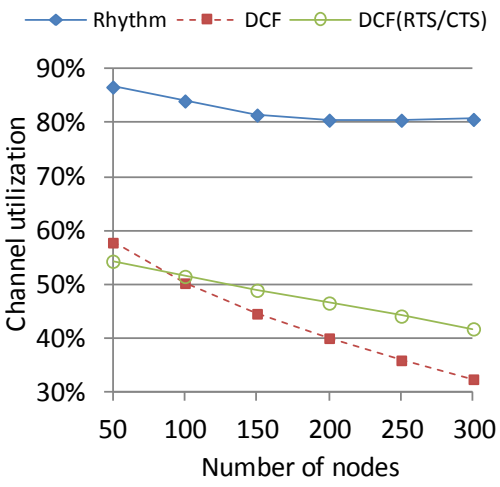
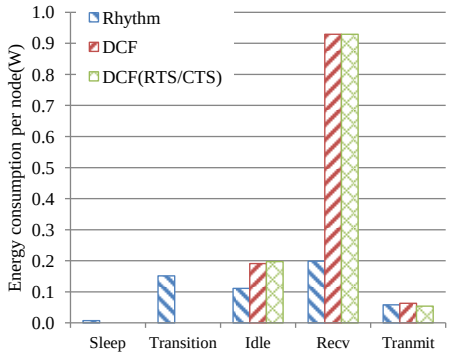
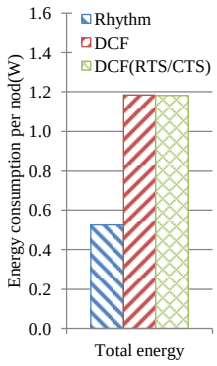


Figure 19: Densely deployed WiFi network

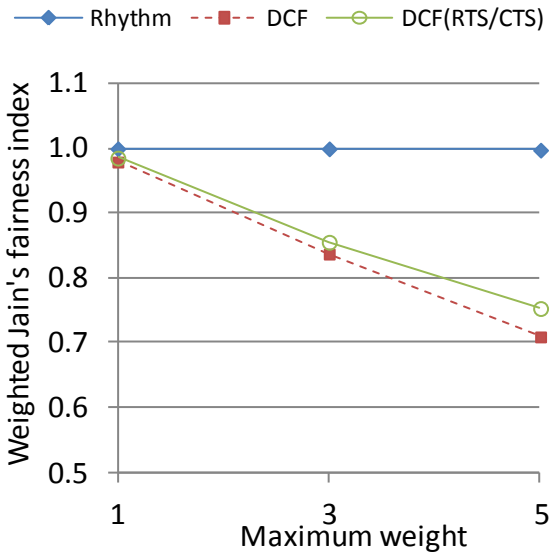


(a) Detail

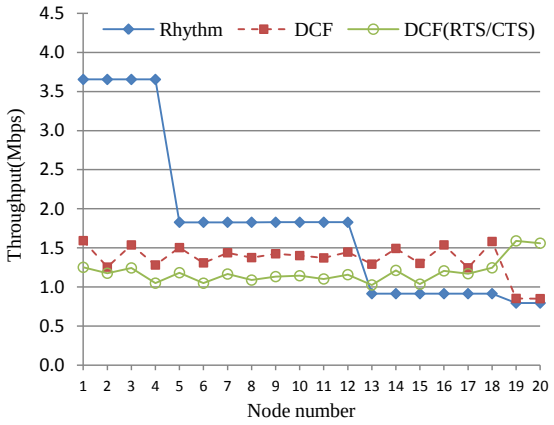


(b) Total Energy

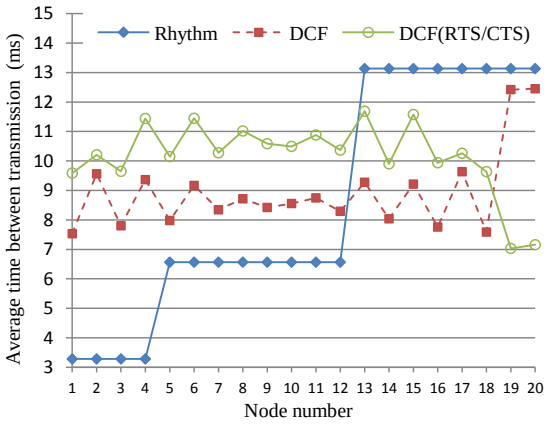
Figure 20: Energy Efficiency of Rhythm



(a) Fairness Index

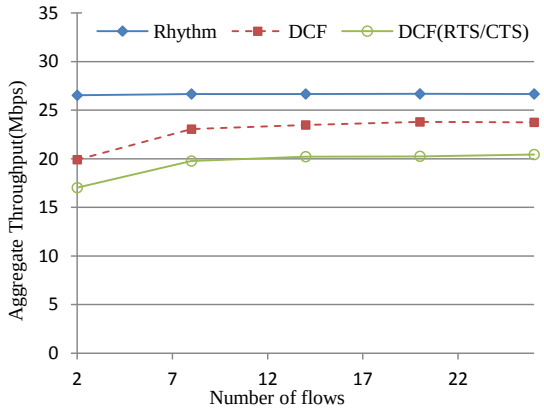


(b) Throughput per Node

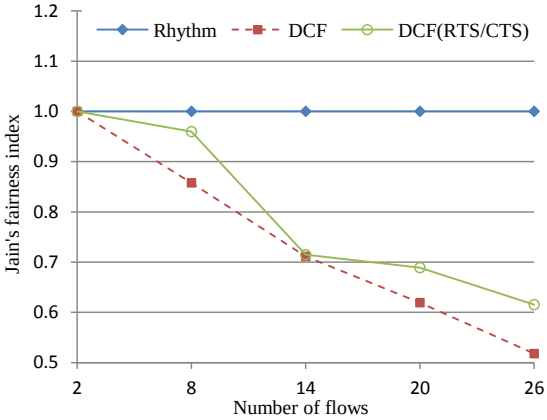


(c) Shortterm Fairness

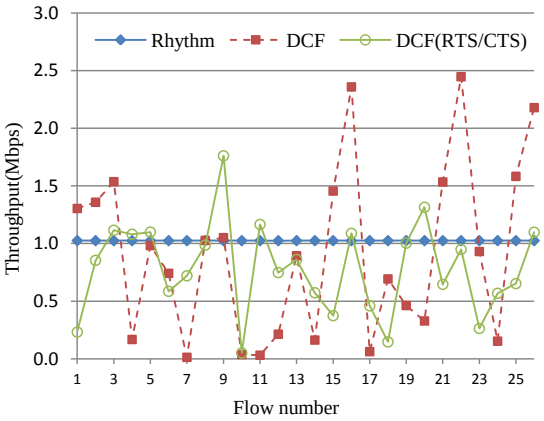
Figure 21: Weighted Fairness of Rhythm



(a) Aggregate throughput



(b) TCP fairness



(c) Throughput for each flow

Figure 22: Protocol awareness of Rhythm

CHAPTER V

BACKWARD COMPATIBILITY OF SCHEDULED WIFI IN FUTURE-PROOFING NETWORKS

5.1 Introduction

In the previous chapter, we proposed Rhythm, which fulfills benefits of central scheduling by achieving *scheduled WiFi*. By bridging between centralized scheduling and purely distributed operations, we answered the following question: Can the goals of centralized scheduling be achieved using purely distributed operations?

However, Rhythm assumes a perfect environment without legacy nodes. Due to the more aggressive contention behavior of nodes operating Rhythm compared to legacy nodes, starvation problems can happen when legacy nodes exist. In this chapter, we present a solution called *Look Who's Talking* (LWT) [62] that achieves *scheduled WiFi* with better *backward compatilibity*. The scope of LWT is restricted to a single collision domain (single or multiple cells), but we briefly discuss how LWT can be extended to multiple collision domains.

5.2 Problem Definition

5.2.1 Scheduled WiFi

Scheduled WiFi is the notion of making WiFi nodes transmit according to an order specified in a target schedule S. Fig. 23 shows a possible system architecture for scheduled WiFi consisting of a central controller and a multi-cell WiFi network. The central controller (which can either be an on-site controller in an enterprise WiFi deployment or a cloud controller for APs that do not belong to the same administrative domain) communicates with

all APs in the WiFi network. Protocols such as CAPWAP([18]) and LWAPP([19]) facilitate the communication between the APs and the central controller. WiFi-related network information is sent to the central controller periodically. Based on the collected information and other configured policies, the central controller determines a target schedule $S = \{s_0, \cdots, s_{k-1}\}$, where $s_i = (tx_i, rx_i)$ indicates the scheduled link in schedule position i (tx_i and rx_i are the MAC addresses of the Tx and Rx of the link). The schedule is then pushed to the nodes in the network through the APs.

We further define a metric called *adherence*, adh, to measure how well the WiFi nodes track the prescribed schedule. The transmission pattern of WiFi nodes is $T = \{t_0, \cdots, t_{m-1}\}$, where t_i can be a successful transmission ((tx_i, rx_i)) or a transmission without ACK (Col). We partition T into several regions $\{T'_0, \cdots, T'_l\}$ using $t_i = Col$ (Fig. 24) as a separator. The adherence is $adh = \frac{1}{m}\Sigma_{i=0}^{l} \max\{HCC(T'_i, S)\}$, where HCC is the hamming cross-correlation function. Note that $adh \geq 0$, and $adh = 1$ means perfect adherence.

The ability to make nodes transmit while following a prescribed schedule has many benefits. Specifically, the benefits center around predictability of service, efficiency under heavy load conditions, and weighted differentiation when applications and services require different resource allocations. More generically, once nodes in a network can be made to follow a schedule, *any MAC problem (e.g. energy-efficient scheduling, transport-layer aware scheduling, usage-based scheduling, etc.) can now be solved in a much easier fashion since only a centralized solution to the problem needs to be constructed*. The output of the centralized scheduler is simply furnished to the network nodes that then achieve that schedule.

5.2.2 Scope and Challenges

The focus of LWT is on WiFi networks with a single channel setting and in a single collision domain, in which any two simultaneous transmissions cause a collision. WiFi network

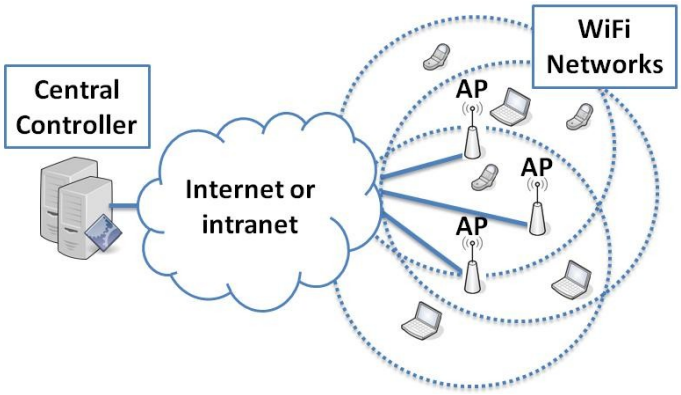


Figure 23: System architecture for scheduled WiFi

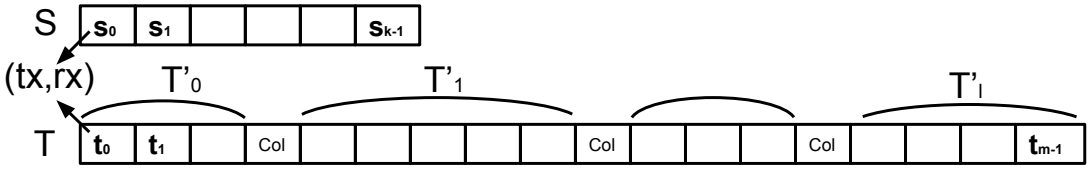


Figure 24: Adherence of a transmission pattern

deployments typically have auto-channel-selection mechanisms (3 to 25 orthogonal channels depending upon the spectrum used). For a given channel, most networks are practically either in a single collision domain or are totally disconnected, and hence can operate independently. We discuss how to extend LWT for multiple collision domains in Section 5.3.5.4 briefly, but leave its in-depth exploration for future work.

Thus, the problem addressed by LWT can be stated as follows. Consider a multi-cell WiFi network containing n nodes in a single collision domain. The central controller decides a target schedule S, which is delivered to all nodes. Having S, how can the nodes achieve scheduled WiFi with minimum overhead? We present below a list of non-trivial challenges that need to be addressed by any scheduled WiFi solution:

Non-backlogged Nodes: A common problem in scheduled WiFi is to deal with non-backlogged nodes. When nodes do not always have packets to transmit, it is hard to determine whether to schedule the node. While collecting queue status from all the nodes is one solution to the problem, this collection incurs non-negligible overheads and delays,

especially when there are a large number of APs or significant delays between the APs and the central controller. Thus, it is desirable to deal with non-backlogged nodes without collecting queue status.

Decodability vs. Detectability: The WiFi PHY layer uses multiple rates for data transmissions. Thus, overheard packets cannot always be decoded correctly. This prevents nodes from fully relying on information from overheard transmissions. To deliver control-plane information, one possible solution is to use additional signals or control frames, which in turn increases overheads. Hence, it is desirable to construct a solution to tackle such non-decodable scenarios while incurring minimal costs.

Partial Connectivity: Due to any partial connectivity caused by network topology or obstacles, hidden terminals exist even in single collision domains. A well-known solution to the hidden terminal problem in WiFi is the exchange of RTS/CTS before data transmissions. However, RTS/CTS introduce considerable overheads. Also, the mechanism cannot solve unfairness problems under certain scenarios (we illustrate this in Section 5.3.4). Thus, it is desirable to achieve scheduled WiFi even in the presence of hidden terminals.

Backward Compatibility: Since it is unrealistic that all devices in a target deployment can be updated, backward compatibility is an important property of any newly designed MAC protocols. Some MAC protocols (e.g., DOMINO [78]) deal with legacy nodes by separating their transmissions into the different period. This introduces a delay for legacy nodes and requires extra control overheads. It is thus desirable to construct a solution that allows legacy nodes to operate normally without additional overheads and delay.

Sleeping Nodes: WiFi radios can be put to sleep to conserve energy. Nodes whose radios were put to sleep need to be able to rejoin the network and sync with the schedule dynamically. A naive solution is to use a fixed duration for each transmission (e.g., DOMINO [78]) so that nodes can use the time elapsed to estimate the current schedule slot after sleeping. However, WiFi supports multiple rates for data transmissions, and packet

sizes can vary for different applications. The fixed transmission times require packet aggregation and fragmentation, which in turn introduces extra delays. Hence, it is desirable to address sleeping while allowing for different transmission durations.

Schedule Changes: It is also important to allow for changes in the target schedule. Unicasting the schedule to each node on any update is reliable but introduces significant overheads. Broadcasting the schedule is efficient but can cause problems if the schedule is not delivered reliably to some nodes. Thus, it is desirable to design a solution that allows for updates to the target schedule.

In the following section, we start with an idealized scenario that does not have several of the above challenges and then progress systematically in presenting a solution that addresses all the challenges.

5.3 LWT: Scheduled-WiFi using Distributed Contention

5.3.1 Baseline algorithm

Consider a simple scenario with a fully connected topology, in which overheard packets can be successfully decoded, and nodes are always backlogged. Under this scenario, we introduce the baseline algorithm of LWT, LWT-Baseline.

5.3.1.1 From Centralized Scheduling to Distributed Contention

Scheduled WiFi is achieved by centralized scheduling protocols through the following steps on each node: i) access to the target schedule, ii) get the current schedule slot, iii) trigger a particular transmission, iv) know when the current transmission ends, v) adjust to the next schedule slot, and get back to step iii). Similarly, LWT achieves scheduled WiFi through the following steps on each node: i) receive the broadcast of the target schedule, ii) use "position synchronization" to get the current schedule slot, iii) reuse backoff mechanism for self-triggering, iv) track transmission completions, v) move the "position pointer" to the next schedule slot, and get back to step iii). Below, we describe the detail of each step in LWT.

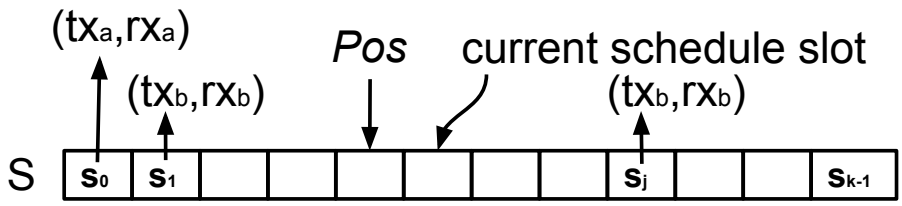


Figure 25: Position synchronization

Algorithm 9 LWT-Baseline

```
1:  function POSSYNC
2:      if (r_0 == Col) or (r_0 == Null) then
3:          ST=RAND, Pos=Null
4:          return
5:      else if ST==RAND then
6:          Pos=PosMatch(-1)
7:      else
8:          Pos=(Pos+1) mod k
9:      end if
10:     if Pos! =Null then
11:         ST=SYNC
12:     end if
13: end function
14: function POSMATCH(prevPos)
15:     i = (prevPos+1) mod k
16:     for j = 1 to k do
17:         if s_i == r_0 then
18:             return i
19:         end if
20:         i = (i + 1) mod k
21:     end for
22:     return Null
23: end function
```

i) *Broadcast of Schedule*: In LWT, APs broadcast the target schedule S to all nodes. S can be put into a beacon every few seconds. Broadcasting generates negligible overheads (the time percentage used to transmit S), which is:

$$\frac{k \times 48 \times 2}{R_b \times T_p},$$

where 48 is the length of a MAC address, k is the length of S, R_b is the sending rate of beacons, and T_p is the period of updating S. Consider R_b = 6Mbps, $k = 100$, and T_p = 1s, the overhead of broadcasting S is only 0.2%. However, broadcasting is unreliable and can cause a problem if the schedule is not reliably delivered to some nodes. We will deal with this issue in Section 5.3.5 when considering the issue of

change to the schedule.

ii) *Position Synchronization*: In order to follow a schedule, nodes need to know the current schedule slot. In LWT, this is achieved by synchronization in a schedule position pointer *Pos*. Each node in the network maintains a schedule pointer *Pos* that points to a position (0 to $k-1$) in the schedule; the next position where *Pos* points to is the current schedule slot (Fig. 25).

Pos synchronization is achieved by overhearing transmissions. Initially, nodes perform random backoff as in DCF. Once a node wins the contention and finishes a transmission successfully, all nodes learn the MAC addresses of the Rx and Tx of this transmission through overhearing. Then, all nodes find *the smallest position of this transmission* in S, and set *Pos* to that position. For example, in Fig. 25, the smallest position of link (tx_b,rx_b) is 1. Matching a link to the smallest position avoids ambiguity when a link is scheduled multiple times [1]. *Pos* synchronization gives all nodes a common current schedule slot. After the first synchronization, it is sufficient to maintain *Pos* synchronization by incrementing the position of *Pos* after each transmission. Note that since DCF requires nodes always to listen to the channel for the backoff mechanism, nodes can track the start and end time of schedule slot. Also, the active listening time of LWT is the same as that of DCF.

Algorithm 9 illustrates the *Pos* synchronization of LWT-Baseline. Assume that a target schedule $S = \{s_0, \cdots, s_{k-1}\}$ is known to all nodes in the network. Nodes record the most recent transmission r_0 (Can be its own transmission or an overheard transmission). $r_0 \in \{s_i\} \bigcup Col$, where Col represents a transmission without ACK. Nodes update *Pos* according to r_0 and a synchronization state *ST*. There are two states of *ST*: *RAND* and *SYNC*. Initially, *ST* is *RAND*. If r_0 is $Null$ (never heard any transmissions) or Col, nodes set *ST* to *RAND* and *Pos* to $Null$. If $r_0 \in \{s_i\}$ and *ST* is

[1] We will talk about how to handle sleep nodes or new nodes in Section 5.3.5.

RAND, which means there is no *Pos* synchronization before this transmission, nodes set *Pos* to the smallest position matching r_0 in S and set *ST* to *SYNC*. If $r_0 \in \{s_i\}$ and *ST* is *SYNC*, nodes simply increment the position of *Pos* by 1 (line 8).

iii) *Self-Triggering*: The backoff mechanism is used to trigger the transmission of the current scheduled link. Note that, since nodes need to freeze the backoff timer once the channel becomes busy, nodes constantly listen to the channel when counting down the backoff number. Thus, once a new transmission is overheard, a node freezes the backoff timer, updates *Pos*, and determines a new backoff number according to the state *ST*. If *ST* is *SYNC*, it uses *Pos* to find the current schedule slot. If the node is scheduled in the current schedule slot, it sets the backoff number to 0. Otherwise, it sets a large backoff number and waits for *Pos* update. If *ST* is *RAND*, the node carries out random backoff as DCF.

iv) *Transmission Completions*: Nodes need to learn the completion of the current transmission to start the next transmission in the schedule. In DCF, nodes always track the completion of each transmission to start the backoff mechanism. LWT utilizes the same mechanism to track transmission completions.

v) *The Next Schedule Slot*: In LWT, nodes go to the next schedule slot simply by incrementing the position of Pos by 1.

5.3.1.2 Efficiency Estimation

We give a brief efficiency estimation of LWT-Baseline by estimating the overhead caused by collisions and packet errors. Assume the success of transmissions when n nodes contend using DCF are independent Bernoulli trials, and P_{col} is the probability of having a collision (failure) for each trial. In LWT-Baseline, one successful transmission achieves *Pos* synchronization and avoids collisions. Thus, the average overhead caused by collision

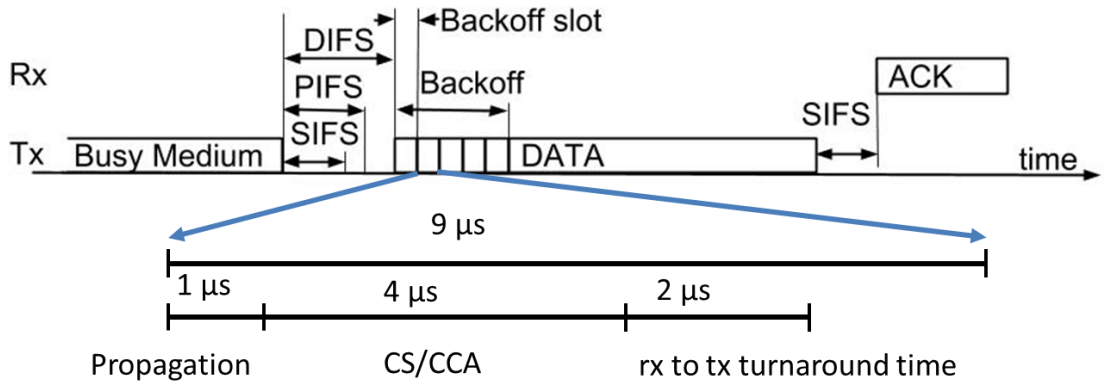


Figure 26: CS/CCA in a backoff slot

in m transmissions is

$$O_{col} = \frac{(P_{col} + \cdots + (m-1)P_{col}^{(m-1)})}{m}(1 - P_{col}) + P_{col}^{(m)} = \frac{\frac{1}{1-P_{col}} - 1}{m}.$$

When $m \to \infty$, $O_{col} \to 0$. That is, the overhead is 0 in long term when there is no packet error. Assume P_{err} is the packet error rate. The overhead caused by packet errors and collisions in DCF is $O_{err\_col} = (1 - P_{col})P_{err} + P_{col}$; this overhead in LWT-Baseline is

$$P_{err} \times (1 - P_{col})(1 + P_{col} \times 2 + P_{col}^2 \times 3 + \cdots) \leq P_{err} \times \frac{1}{1 - P_{col}}.$$

Considering $P_{err} = 1\%$ and $P_{col} = 15\%$, $O_{err\_col} = 15.85\%$ in DCF, and $O_{err\_col} \leq 1.18\%$ in LWT-Baseline.

5.3.1.3 Limitations of Baseline algorithm

Even though LWT-Baseline achieves scheduled WiFi efficiently, it is limited to simple scenarios where nodes are always backlogged, and transmissions can always be overheard and successfully decoded. As mentioned in Section 5.2, we will consider other practical issues sequentially in the following subsections.

5.3.2 Work conservation with non-backlogged nodes

In this section, we consider scenarios with non-backlogged nodes. As mentioned in Section 5.2, it is hard to determine the schedule when nodes do not always transmit. However,

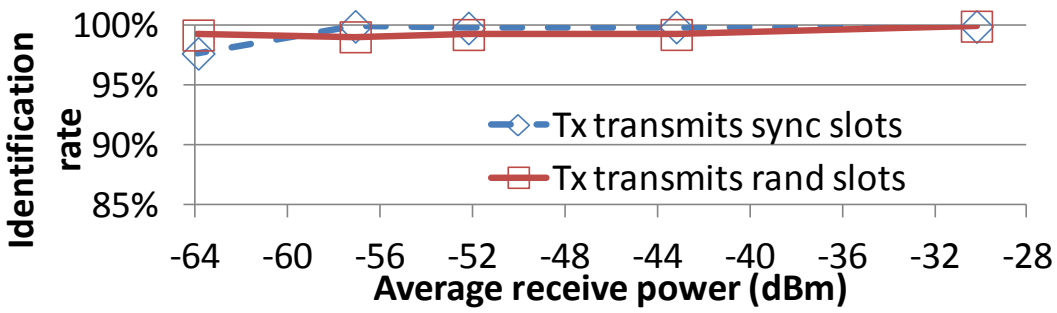


Figure 27: Identification rate of different slots

collecting queue status from all nodes brings overheads and delays. We introduce a *work-conserving* mechanism in LWT (LWT-WC), which utilizes the carrier sense/clear channel assessment (CS/CCA) mechanism of WiFi to automatically release available resources to other nodes when the scheduled node doesn't transmit.

5.3.2.1 CS/CCA of WiFi

The core insight of LWT-WC lies in the CS/CCA mechanism of WiFi. WiFi packets contain a PLCP preamble in the beginning for synchronization. It is also used for CS/CCA to identify the start of a transmission through auto-correlation or cross-correlation. The IEEE 802.11 standard specifies that an OFDM transmission at a receive level no less than the minimum sensitivity (-82dBm for 20MHz channel) shall cause CS/CCA to indicate channel busy with a probability over 90% within 4 μs (for 20MHz channel)[20]. This timing is sufficient for nodes to identify the start of a transmission within a backoff slot (9 μs). Fig. 26 gives a detailed illustration. A node starts to transmit at the beginning of a backoff slot if its backoff timer expires. The propagation delay is around 1 μs, and nodes take 4 μs to identify the start of a transmission. If a transmission is detected, nodes hold the backoff timer. If no transmission is detected, since the receive to transmit turnaround time of a node is less than 2 μs [20], it is sufficient for a node to prepare to transmit at the beginning of the next backoff slot if its backoff timer expires.

Algorithm 10 LWT-WC

```
1: function POSSYNC
2:     if (r_0 == Col and w_t < w_t_thd) or (r_0 == Null) then
3:         % rand slot does not trigger reset of ST
4:         ST=RAND, Pos=Null
5:         return
6:     else if ST==RAND then
7:         Pos=PosMatch(-1)
8:     else
9:         Pos=(Pos+1) mod k
10:    end if
11:    if Pos! = Null then
12:        ST=SYNC
13:    end if
14: end function
15: function SELFTRIGGER
16:    if ST==SYNC and myself==current scheduled node then
17:        set backoff to zero
18:    else if ST==SYNC then
19:        do random backoff as DCF with b ≥ 1
20:    else
21:        do random backoff as DCF
22:    end if
23: end function
```

5.3.2.2 Work-Conserving

Utilizing the ability of identifying the start of a transmission within a backoff slot (which is an ability required by IEEE 802.11 standard [20]), LWT-WC classifies schedule slots into different types according to the timing of the start of the transmission.

Classifying Schedule Slots: LWT-WC classifies slots into two types: "sync slot" and "rand slot." If the current scheduled node transmits, it is identified as a sync slot. Otherwise, this slot becomes a rand slot, and all nodes can contend for this slot using DCF. The identification of sync slot and rand slot is automatically carried out with CS/CCA and backoff mechanism. In LWT-WC, when nodes are not scheduled in the current schedule slot, they pick a random backoff number ≥ 1. Since the current scheduled node sets backoff number to zero, nodes identify the start of a transmission within the first backoff slot if the current scheduled node transmits. Otherwise, if there is no transmission starting within the first backoff slot, nodes continue to backoff and this slot becomes a rand slot.

5.3.2.3 Experimental Validation of Work-Conserving

We set up WiFi communications using WARP [15] to validate the core insight of LWT-WC. In our experiment, one node acts as Tx and transmits unicast packets to the other node, which acts as Rx and tries to figure out which slot the Tx uses. We evaluate slot identification rate when the Tx always transmits in sync slots and when the Tx always transmits in rand slots. When transmitting in rand slots, the Tx sets backoff number to one, which creates the smallest time difference between rand slots and sync slots. Fig. 27 shows the correct identification rate of both scenarios under different receive power. The experiment result indicates that it is reliable (over 98% correctness) to use CS/CCA to identify a sync slot and a rand slot. In Fig. 27, the identification rate when Tx transmits in sync slot goes down to 98% when the received power is -64dBm. It is because that the probability of CS/CCA correctly indicating channel busy becomes lower in low receive power. If CS/CCA fails to identify the transmission in the first backoff slot, the sync slot is wrongly identified as a rand slot.

5.3.2.4 Algorithm

Algorithm 10 illustrates the Pos synchronization and self-triggering mechanism of LWT-WC. Nodes record a parameter w_t, which is the waiting time between the current transmission and the previous transmission, and a parameter w_{t_thd}. If the previous transmission is Col, w_{t_thd} is set to (EIFS $+SlotTime$/2), where EIFS is the interframe space after an erroneous transmission [20] and $SlotTime$ is the time duration of a backoff slot; otherwise, w_{t_thd} is set to (DIFS $+SlotTime$/2). Transmissions in rand slots won't trigger reset of ST even if it is a collision Col. That is, if there is a collision in a rand slot, ST will remain the same value, and if *ST* is *SYNC*, nodes simply increase Pos by one (line 9). When nodes are not scheduled in the current schedule slot, they pick a random backoff number $b \geq 1$.

5.3.2.5 *Efficiency Estimation*

Implementation of work-conserving mechanism can increase the overhead of LWT since the rand slots can have larger backoff slots and collisions. As estimated in Section 5.3.1.2, the overhead caused by packet errors and collisions in DCF is

$$O_{err\_col} = (1 - P_{col})P_{err} + P_{col}.$$

This overhead in LWT-Baseline is not larger than $P_{err} \times \frac{1}{1-P_{col}}$. The overhead of LWT-WC is the linear combination of the overhead of DCF and LWT-Baseline, weighted by the time portion of the rand slots and sync slots. When the traffic load is high, the channel efficiency of LWT-WC approaches that of LWT-Baseline, which is near-optimal. When the traffic load is low, the channel efficiency of LWT-WC approaches that of DCF, which is efficient enough under low traffic load.

5.3.3 Decodability vs. Detectability

In this section, we consider scenarios in which overheard transmissions cannot be decoded. As mentioned in Section 5.2, nodes sometimes can not get information from overheard transmissions. However, using extra signals/control frames introduces overheads. Although nodes in LWT only need the Tx and Rx addresses of the first successful transmission for Pos synchronization, resynchronization can be triggered by the change of schedule, sleeping nodes, new nodes, transmission errors, and collisions. Identifying the addresses of transmissions in sync slots is important under those re-synchronization situations. Thus, we introduce a conditional Viterbi algorithm in LWT (LWT-CV) to help nodes identify addresses of a transmission when it cannot be decoded.

5.3.3.1 *Theoretical Error Rate of Identifying Addresses*

The core insight of conditional Viterbi algorithm is: *instead of considering all decode paths, considering only a subset of decode paths when decoding*[2]. Since the target schedule

[2]Please find the definition of decode path in Chapter 10.5 of [37]

S contains addresses of all nodes in the network, we can consider only the n addresses in S rather than all MAC addresses. This reduces the number of decode paths from 2.8×10^{14} to n (the number of nodes), which practically is less than 50 in a single collision domain. The significant reduction of the number of decode paths substantially increases the distance between paths and also restricts the possibility of errors to only certain bits. Thus, the error rate decreases significantly.

5.3.3.2 Conditional Viterbi Algorithm

- *Convolutional Code*: WiFi uses the convolutional code for channel coding. Convolutional code [37] is an error-correcting code . Small bit errors can be recovered by observing the whole sequence. The encoder can be viewed as a finite-state machine.

- *Decoding with an Address Tree*: Viterbi algorithm is an optimal algorithm for decoding convolutional codes. It utilizes dynamic programming to find the decode path with minimum distance. Conditional Viterbi algorithm is based on Viterbi algorithm, but only considers certain decode paths. An address tree is built according to addresses in S, and only the decode paths in the address tree are considered.

5.3.3.3 Algorithm

Algorithm 11 illustrates the conditional Viterbi Algorithm, where $tree$ is the address tree built according to addresses in S. Each time the received/overheard transmission can not be decoded, conditional Viterbi algorithm is used to identify the Rx and Tx addresses. A parameter, $tree\_state$, is used to trace the current location in the address tree. $tree\_state$ is set to the root of address tree at the first bit of each address (line 12). Then, only the input that has a path in the tree will be considered when decoding (line 15).

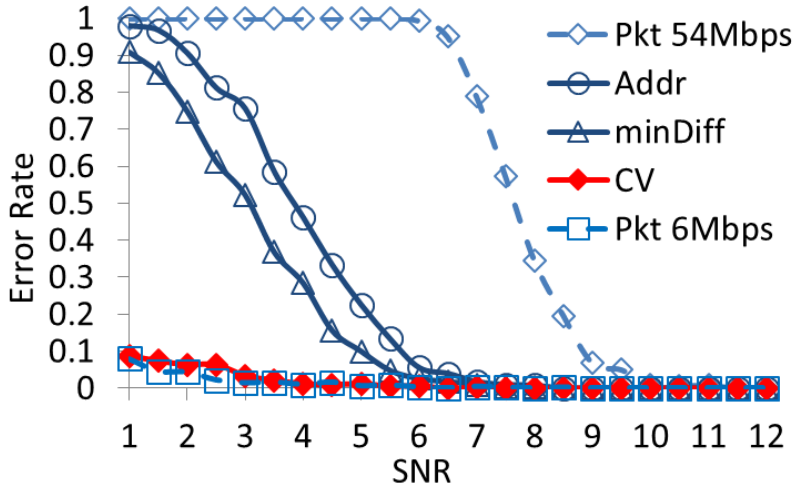
5.3.3.4 Evaluation of Conditional Viterbi Algorithm

We use Matlab simulation to evaluate conditional Viterbi Algorithm. In the simulation, m MAC addresses are randomly generated. For each of the m addresses, another address

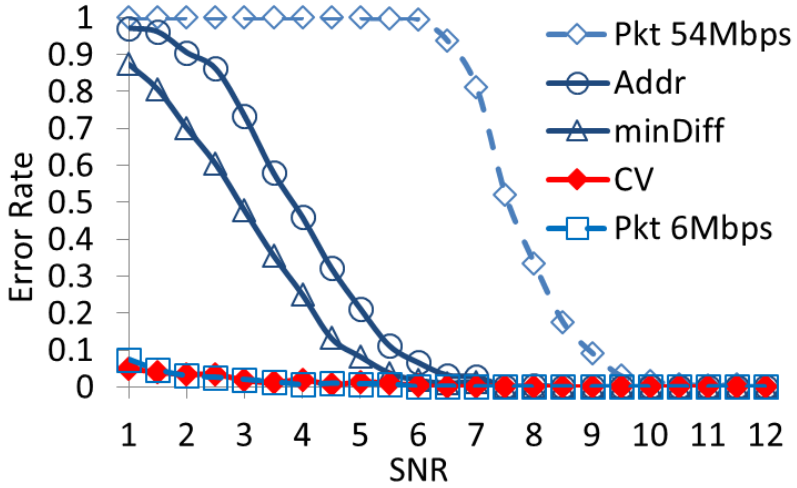
Algorithm 11 Conditional Viterbi Algorithm

```
1: function CONDVIT(recv_bits, trellis, end_of_addr, init_of_addr, tree)
2:     NumStates= trellis.numStates
3:     n_addr_bit = 48
4:     metrics = zeros(NumStates, end_of_addr+1)+inf
5:     path = zeros(NumStates, end_of_addr+1)-1
6:     tree_state = zeros(NumStates, end_of_addr+1)-1
7:     metrics(1,1)=0
8:     for i = 1 to end_of_addr do
9:         for  state = 1 to NumStates do
10:            if metrics(state,i)!=inf then
11:                if (i==init_of_addr)or(i== init_of_addr + n_addr_bit) then
12:                    tree_state(state,i)=root
13:                end if
14:                for  input = 0 to 1 do
15:                    if (i<init_of_addr)or(tree(tree_state(state,i),input)!=NULL) then
16:                        next_state=trellis.nextStates(state,input)
17:                        output= trellis.outputsstate,input
18:                        mt=sum_diff(output,recv_bits(i))+metrics(state,i)
19:                        if mt < metrics(next_state,i+1) then
20:                            metrics(next_state,i+1)= mt
21:                            path(next_state,i+1)= state
22:                            if i ≥ init_bit_addr then
23:                                next_tree_state=tree(tree_state(state,i),input)
24:                                tree_state(next_state,i+1)=next_tree_state
25:                            end if
26:                        end if
27:                    end if
28:                end for
29:            end if
30:        end for
31:    end for
32:    % trace back the path with minimum metrics
33:    decode = TraceBackPath(path, min(metrics), trellis)
34:    decoded_addr=decode(init_of_addr:end_of_addr)
35: end function
```

that has different last 4 bits to it is generated. (Note that for each address, there is another address having the same first 44 bits.) From the $2m$ addresses, two addresses are randomly selected. Transmissions containing the two addresses and random payload are received though an AWGN channel with different SNR. Fig. 28(a) and 28(b) shows the address identification error rate of conditional Viterbi algorithm when the transmission rate is 54Mbps (64-QAM with 3/4 coding rate) with $2m = 50$ and $2m = 20$. Only the least robust rate is shown to keep the graph clear. The identification error rate of other rates is smaller than that of 54Mbps. The packet error rate of 6Mbps (Pkt 6Mbps) and 54Mbps



(a) $2m$=50 possible addresses



(b) $2m$=20 possible addresses

Figure 28: Address identification error rate

(Pkt 54Mbps) are also shown as references. The error rate of choosing the addresses in S that has the minimum distance to the decoded addresses of Viterbi algorithm, which is the "minDiff" in figures, is also presented. As indicated in figures, conditional Viterbi algorithm significantly decreases the identification error rate of addresses.

5.3.4 Partial Connectivity

In this section, we consider scenarios in which some transmissions cannot be overheard, causing hidden terminals problems. RTS/CTS is a famous solution for hidden terminal

problems. However, it introduces considerable overheads and has unfairness problems in certain scenarios. Thus, we propose a transparent transmissions mechanism in LWT (LWT-TT) to deal with hidden terminal problems in a single collision domain.

5.3.4.1 Hidden Terminal Problem in a Single Collision Domain

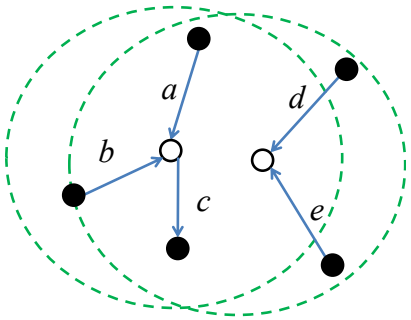
We define a single collision domain as "a set of links in which transmissions through any two of them cause a collision." Fig. 29(a) shows a single collision domain. Hidden terminal problems happen when Tx of links cannot be overheard. We classify hidden terminal problems in a single collision domain into two types:

i) *HtRx*: At least one node of each link can be overheard by all nodes in the single collision domain.

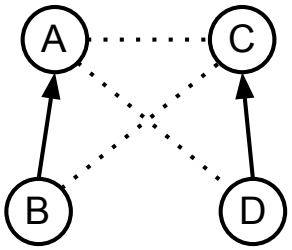
ii) *HtNRx*: There exists a link where both Rx and Tx cannot be overheard by some nodes in the single collision domain. Fig. 29(b) and 29(c) show examples of both scenarios.

A well-known solution for hidden terminal is exchanging RTS/CTS before a data transmission. However, RTS/CTS introduces extra 116 μs to each transmission. Consider a packet of 1500bytes transmitting in 54Mbps, the overhead caused by RTS/CTS is 26%. Also, RTS/CTS can not solve unfairness problems in HtNRx scenarios. In Fig. 29(c), it is very hard for node D to win a contention. Since the transmission of node B interferes the reception of node C, but the transmission of node D does not interfere reception of node A or B, RTS transmitted from node D to node C can be corrupted by node A or B easily. Also, since node D can not hear the RTS/CTS of node A and B, it can keep re-transmitting RTS while any of the two nodes are transmitting. This makes node D use unnecessary large backoff numbers.

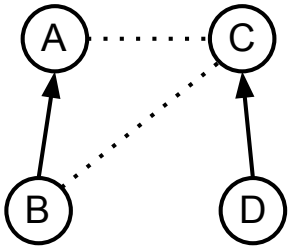
For LWT to gain *Pos* synchronization, the start and end time of a transmission need to be learned by all nodes in the single collision domain. We first declare that "if every



(a) Single Collision Domain



(b) HtRx



(c) HtNRx

Figure 29: Examples of hidden terminals in a single collision domain

node that hears a transmission broadcasts the information of this transmission, all nodes in a single collision domain can get the information of this transmission." The proof of this statement is simple. If there is a node that can not get the information of a transmission, this means that all of its Rx cannot hear this transmission. Thus, this node actually can transmit without interfering with the on-going transmission, which contradicts the definition of a single collision domain. Below, we use this statement and introduce the transparent transmissions mechanism.

5.3.4.2 *Transparent Transmissions*

To pass the transmission information to all nodes in the single collision domain, we propose a mechanism named *transparent transmissions*. When a node receives or overhears a transmission, it transmits a special signal, tt_Start, to indicate the start of a transmission. After the transmission is finished, the node transmits during SIFS another special signal, tt_End, indicating the end of this transmission. Fig. 30 illustrates the timeline of a transparent transmission. Nodes receive tt_Start and tt_End can use the timing of these signals to estimate the start and end time of the current transmission. Since all nodes that hear the current transmission transmit those signals, all nodes in the single collision domain learn the start and end time of the current transmission. Transparent transmissions needs to satisfy the following conditions:

i) When overlapping with data packets, the special signal can be easily identified under low SNR without extra hardware requirements.

ii) The transmission of the special signal can not disturb the reception of data packets.

iii) The reception of a special signal can not disturb the reception of data packets.

- *Flash Signal*: We use flash signals introduced by Flashback [27] as the special signal in transparent transmissions [3]. Flash signals are sinusoids with a frequency equal to a certain subcarrier of the current WiFi channel, and with a duration equal to an OFDM symbol (4μs). Since flash signals can use any of the 36 subcarriers in a WiFi channel, there are at least 36 different flash signals, and transparent transmissions only use 2 of them for tt_Start and tt_End. Since the Fourier transform of a sinusoidal wave is delta functions, as evaluated in the experiments of Flashback [27], flash signals can be detected easily by simple peak detection algorithm without extra

[3] Transparent transmissions can work with other special signals such as Gold codes [35] used by DOMINO [78] or the correlatable symbol sequences in 802.11ec [51], as long as the signals can be easily identified. Flash signals are selected due to its simplicity.

hardware requirements (the transmit/detect of flash signals can be achieved through software/firmware changes by reusing OFDM DSP blocks.). The ease of detection guarantees the robustness of transparent transmissions. Still, packet receptions won't be disturbed by flash signals as long as the flash rate is less than 50,000 flashes per second. Note that a flash signal can contain multiple sinusoids, each with a different frequency (subcarrier). Increasing the number of subcarriers improves the robustness and detection rate of a flash signal. For example, consider a situation when a node receives two flash signals from different transmitters simultaneously. If containing only one subcarrier, the two flash signals may have a destructive effect on each other and cannot be detected by the receiving node. This situation can be avoided by having multiple subcarriers in a flash signal. The probability of destructive effect drops exponentially as the number of subcarriers increases.

- *Switch to Transmit while Receiving*: Due to the half-duplex property of WiFi radios, a node can not transmit a flash signal while receiving. Thus, we introduce a "switch to transmit while receiving" mechanism for nodes to transmit flash signals while receiving. A tt_padding, which contains only zeros, is inserted after the PHY header of each packet. After receiving the PHY header of a packet, a node switches to transmit tt_Start, and then switch back to continue receiving. Since the received portion that missed due to transmitting tt_Start is tt_padding, the packet can be received successfully. tt_padding also ensures that the reception of special signal does not disturb the reception of data packets. The length of tt_padding considers both switch time and signal transmission time. As will be shown in the experimental results in Section 5.3.4.3, we select the length of tt_padding based on the time synchronization requirement of IEEE 802.11 [20].

- *Transmit during SIFS*: To inform other nodes the end of the current transmission, another flash signal, tt_End, is transmitted after the DATA transmission is finished.

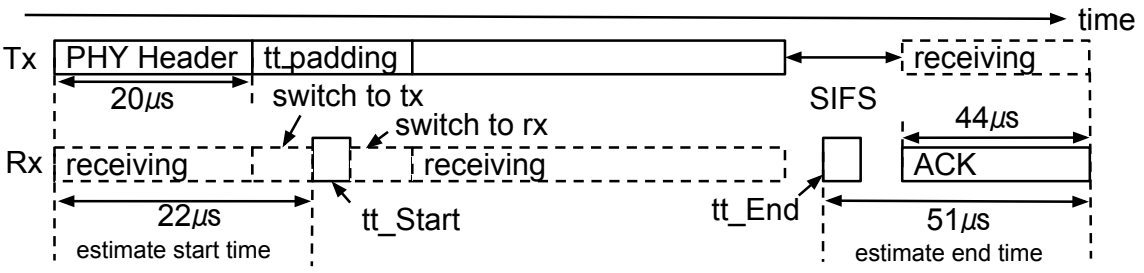


Figure 30: Timeline of transparent transmissions

Nodes utilize the SIFS duration between DATA and ACK to transmit tt_End. Since SIFS is 10μs, the duration of tt_End is 4μs, and the receive to transmit turnaround time is smaller than 2μs, the time duration of SIFS is quite sufficient for transmitting tt_End.

5.3.4.3 Experimental Validation of Transparent Transmissions

Experiments using WARP [15] are carried out to validate transparent transmissions. In the experiment, a Tx generates a preamble following random data bits using 300 OFDM symbols. After receiving the preamble from the Tx, a Rx switches to transmit mode and transmits a sine wave (represents tt_Start) for 4μs. After this transmission, the Rx switches back to receive mode and keep receiving. Fig. 31 shows a received signal wave with two different scales; the unit of the x-axis is 2.5e-8 seconds. There is a small duration (around 8μs) of "no received signal" when the Rx switches to transmit. Table 7 shows the summary of the results. The duration of "switch to transmit" measured by WARP is: (Avg: 8.7μs, Max: 9μs, Min: 8μs). Since the duration of one OFDM symbol is 4μs, "switch to transmit" usually damages reception of 3 OFDM symbols. The total number of error symbols in the 300 symbols is: (Avg: 3.2, Max: 5, Min: 2). Since the symbol error can be caused by channel noise, it is more accurate to estimate the damage using time duration. According to the experiment results, we use 3 OFDM symbols (12μs) for tt_padding.

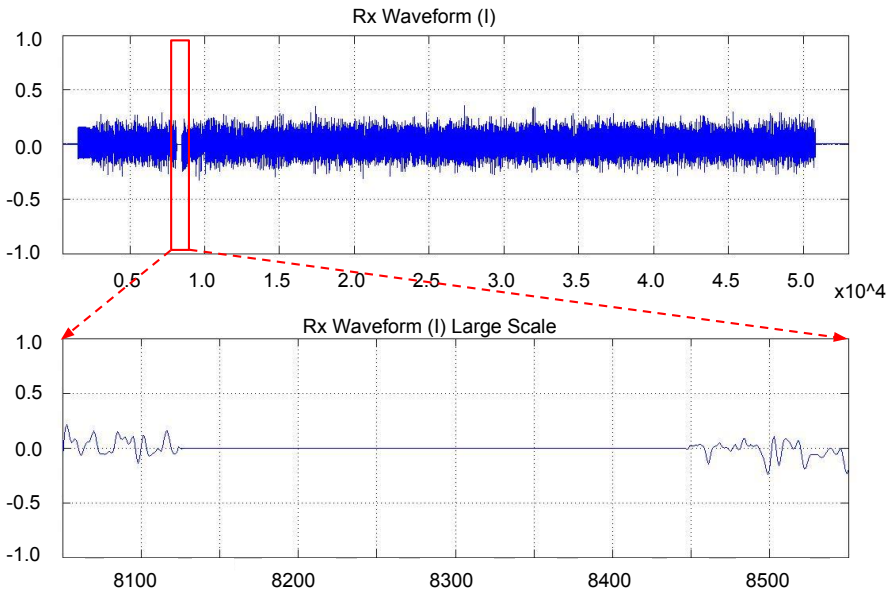


Figure 31: Rx wave of transparent transmissions

Table 7: Evaluation of Transparent Transmission

Measurement	Avg	Max	Min
Duration of switching and transmitting	8.7μs	9μs	8μs
Total Number of Error Symbols	3.2/300	5/300	2/300

5.3.4.4 Algorithm

The *Pos* synchronization and self-triggering of LWT-TT is similar to that of LWT-WC, only with small differences.

For self-triggering, every node maintains a record of "overheard nodes." Initially, a node assumes that it can not overhear any node and update this record as it starts to overhear transmissions. If the Tx of current schedule slot is recorded as "can not overhear," the node waits for extra 24μs after DIFS for possible flash signals. Otherwise, the self-triggering is the same as that of LWT-WC.

For *Pos* synchronization, since transparent transmissions only carry information about the start and end time of a transmission, there are two unknown information: i) if ACK is transmitted, and ii) the Rx and Tx addresses of the transmission. Again, nodes utilize the "overheard nodes" record. If the Rx of the transmission cannot be heard, nodes assume that

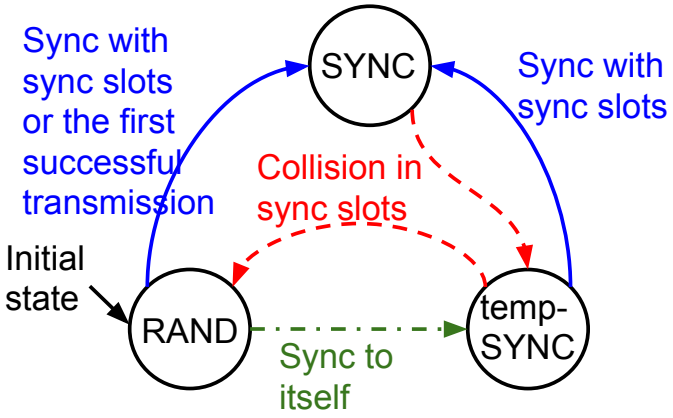


Figure 32: State diagram of synchronization state *ST*

ACK is transmitted and will not reset its *ST* to *RAND*. If the Tx of the transmission cannot be heard, nodes stay in *RAND* when *ST* is *RAND*, and increase *Pos* by 1 if *ST* is *SYNC*. (Note that nodes still can use timing to identify sync slots and rand slots.)

A new state of *ST*, *tempSYNC*, is introduced for ease of *Pos* synchronization. If nodes hear sync slots while its *ST* is *RAND*, it tries to synchronize using those sync slots. However, if the number of overheard sync slots is not enough for a node to figure out the position of Pos, a node can synchronize to itself and switches its *ST* from *RAND* to *tempSYNC* when it has the first successful transmission in a rand slot. When *ST* is *tempSYNC*, nodes setup its *Pos* (match to its own successful transmission) and transmit in its scheduled slots indicated by its *Pos*. If the transmissions (transmit using sync slots) in the scheduled slots succeed, nodes switch its *ST* from *tempSYNC* to *SYNC* after a schedule duration. Otherwise, if any collision happens in the sync slots, *ST* is switched from *tempSYNC* to *RAND*.

On the other hand, if a collision happens in a sync slot, a node will not directly switch from *SYNC* to *RAND*. It first switches from *SYNC* to *tempSYNC* for a schedule duration. If a collision happens in a sync slot again when *ST* is *tempSYNC*, it switches from *tempSYNC* to *RAND*. Fig. 32 shows the state diagram of *ST*. The rest of the algorithm is same as that of LWT-WC.

5.3.4.5 *Efficiency Estimation*

The protocol overhead of transparent transmissions is the extra time duration of tt_padding in a packet. Assuming a 1500byte packet is transmitted in 54Mbps, the overhead caused by tt_padding (12μs) is only 3%, and this overhead decreases as the transmission rate become lower. Comparing to the 26% overhead caused by RTS/CTS, transparent transmissions is much more efficient.

5.3.5 Other challenges and considerations

5.3.5.1 *Supporting backward compatibility*

LWT is backward compatible and can operate with legacy nodes without separating them to a different time duration. Legacy nodes can transmit in rand slots, in which all nodes run DCF. If the traffic load from legacy nodes is high, the central controller can schedule special Legacy-only slots that LWT nodes won't contend. Although the probability of legacy nodes disturbing sync slots is non-zero, this probability decrease exponentially due to the exponential grows of the contention window (cw) in DCF. For example, since the probability of a legacy node selecting zero as its backoff number with cw=15 is 6%, the probability of a legacy node disturbing a sync slot is 6% in the first time. This probability becomes 3% (cw=31) for the second time and less than 1.5% (cw$\geq$63) after the third time. The *tempSYNC* state also helps LWT nodes to be more robust against disturbance.

If there are legacy nodes that can not overhear the current transmission, the Tx will use a tt_padding of 13 OFDM symbols and the Rs transmits CTS (44 μs) with tt_Start during the tt_padding. The disturbing probability, which also decreases exponentially, is 18% in this situation (the probability of a legacy node selecting a backoff number $\leq$2, which leads to a backoff time $\leq 22\mu$s).

LWT can also use PIFS, an IFS shorter than DIFS, in sync slot to decrease the disturbing probability. PIFS is used by APs and non-AP nodes under PCF, in which the behavior of non-AP nodes is under control of the AP. Since the behavior of nodes in sync slots is also

under control of the central controller, it is reasonable to allow nodes to use PIFS in sync slots.

In LWT, nodes need information of all MAC addresses in the network for LWT-CV. Since all legacy nodes need to associate with an AP, the MAC addresses of legacy nodes can be collected by APs and give to all nodes the same way as the target schedule S. Nodes can also collect legacy MAC addresses by learning from overhearing.

5.3.5.2 Sleeping nodes

LWT nodes lose *Pos* synchronization after sleeping since they cannot overhear transmissions during sleeping. As mentioned in Section 5.2, using fixed duration for each transmission can make sleeping nodes keep *Pos* synchronization. However, fixed transmission time requires packet aggregation and fragmentation, which introduces extra delays. Thus, instead of using fixed duration for each transmission, sleeping nodes achieve *Pos* synchronization by overhearing sync slots or synchronizing to itself through *tempSYNC* state (Fig. 32). Once a node wakes up from sleeping, it operates DCF until it hears enough sync slots to identify the position of *Pos*. However, if the node didn't hear any sync slots for a schedule duration, it switches to *tempSYNC* state when it successfully finishes a transmission in a rand slot (synchronizes to itself). If the later transmissions in sync slots succeed, nodes switch to *SYNC* state after a schedule duration. Otherwise, if any collision happens in sync slots, *ST* is switched from *tempSYNC* to *RAND*.

5.3.5.3 Schedule updates and membership changes

The target schedule is determined by the central controller using scheduling algorithms such as [32]. The target schedule is broadcasted by APs periodically. Although broadcasting is efficient, it is not reliable. Transmitting following different schedules can cause collisions in sync slots. Thus, LWT requires a mechanism that quickly indicates the update of S. The update of S is indicated by another flash signal, S_update, transmitted by APs along with DATA or ACK. Since flash signals can frequently be transmitted (50,000

flashes per second) without disturbing receptions and can be identified even when colliding with other flash signals, nodes learn the update of S quickly and reliably. Once receive S_update, nodes operate DCF until they overhear the new S. Thus, although there might be some delay in getting the new S, nodes won't disturb sync slots once they receive S_update.

When a node wants to join the network, it operates DCF until overhearing S from APs. New nodes get Pos synchronization the same way as sleeping nodes. Each AP maintains a client list and periodically sent it to the central controller for updating S.

5.3.5.4 Multiple collision domains

So far, we propose algorithms for WiFi networks in a single collision domain. It is possible to extend them for multiple collision domains, but it is beyond the scope of this thesis. The main idea is using different flash signals to delivery transmission information of different collision domains. There are 36 subcarriers that can be used by flash signals and LWT only needs 2 for each collision domain. In multiple collision domains, multiple transmissions can be scheduled in the current schedule slot. Nodes increase Pos by one only after confirming the end of transmissions of all collision domains in the current schedule slot. This can be learned by either overhearing or receiving flash signals. We leave the in-depth exploration of multiple collision domains for future work.

5.3.5.5 Other WiFi technologies

There have been considerable advances in WiFi technologies from the original 802.11 standard to the pre-ratified 802.11ac standard, and more recently to the ongoing effort to standardize 802.11ad[33, 44, 38] in the 60GHz band. These different advances happen to be in terms of improving physical layer capacities by adapting operating frequency, bandwidth, signal processing algorithms, antenna technologies, coding strategies, and modulation techniques. LWT, on the other hand, adapts the WiFi *coordination mechanism* and hence **can be applied to and used with any of the above 802.11 standards**. We provided discussions of combining scheduled WiFi and MIMO technologies in the future work section.

Table 8: Throughput comparison of LWT and DCF in experiments

Link Number	LWT(Mbps)	DCF(Mbps)
1	9.00	6.62
2	8.03	8.98
3	9.00	6.62
4	8.03	9.00
Total	34.16	31.22

Table 9: ns-3 parameters

Parameter	Value
Frame size	1500byte
Basic transmission rate	6Mbps
Data transmission rate	54Mbps
Slot time	9μs
SIFS	10μs
DIFS	28μs
AIFS	37μs
RX/TX switching delay	$<2\mu$s

5.4 Evaluation

In this section, we evaluate LWT using real-time experiments and ns-3 simulations [10].

5.4.1 Experimental Evaluation

We implement LWT in a software defined radio platform: Wireless open-Access Research Platform (WARP) v3. WARP supports modification and monitoring of parameters and functions in both the MAC and PHY layer and can operate fast enough to perform WiFi communications with off-the-shelf WiFi devices.

As shown in Fig. 11, we set up 3 WARP nodes in a fully connected topology. One acts as AP and the other 2 act as STAs. The experiment is set up in an indoor environment using 802.11a in 5.18 GHz channel with the 54Mbps data transmission rate. We use iperf to generate traffic on the two uplinks and two downlinks. Each link is scheduled once in S.

Table 8 shows the throughput of LWT and DCF from experimental evaluation. Comparing to DCF, LWT gives better throughput and fairness. The performance of LWT is very similar to that of Rhythm (Table 4). Due to the low contention (only three nodes), the difference in throughput is not significant.

Although there are only three nodes in the WARP-based evaluation, it proves that:

(i) the schedule tracking, slot start/end time tracking, and CS/CCA mechanisms all work harmonically under LWT design to achieve better system throughput and fairness.

(ii) This implementation is done without special time synchronization mechanism or hardware change support, which proves that LWT requires only the time synchronization level of 802.11 standard and can be implemented with only software/firmware changes.

We would also like to point out that other features of LWT, such as slot identifications and Transparent transmissions, though not be evaluated under the three-node experiment setting, have been evaluated through other WARP experiments in previous sections.

We take these experiment results as a proof of concept of implementing LWT and evaluate the performance of LWT in more complicated situations using ns-3 simulations.

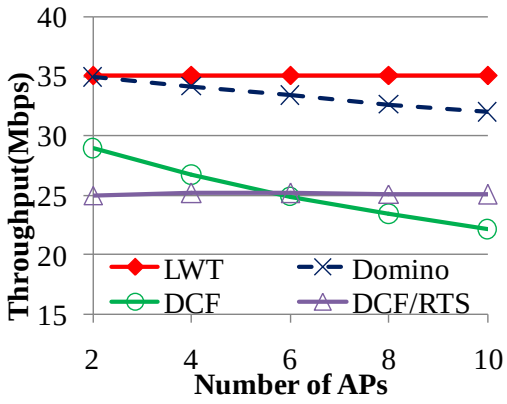
5.4.2 Simulation Based Evaluation

We present the performance evaluation of LWT, DOMINO, DCF, and DCF with RTS/CTS (DCF/RTS) using ns-3 simulation. Table 9 shows the simulation parameters, which follows 802.11g. In all scenarios, traffic is generated on all uplinks and downlinks, and each link is scheduled once in S. In all situations, the theoretical optimal throughput is 36Mbps.

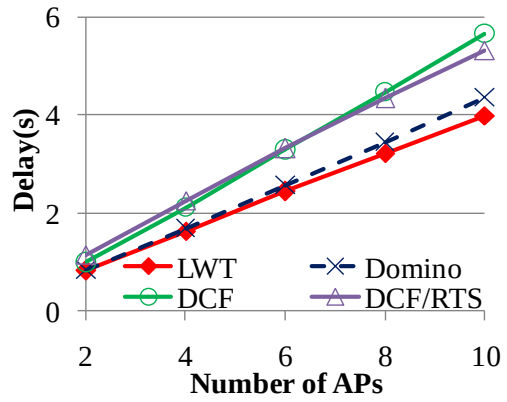
We first evaluate the performance of each mechanism under fully connected topology (no exposed terminals and no hidden terminals). Scenarios with and without non-backlogged nodes are both considered. Then, the performance of each mechanism is evaluated under the two types of hidden terminal scenarios: HtRx and HtNRx.

5.4.2.1 Saturated traffic in fully connected topologies

We set up fully connected topologies with saturated traffic using a different number of APs. Each AP has two clients. We consider a different number of nodes since there can be a significant diversity in node density in practice.



(a) Network throughput

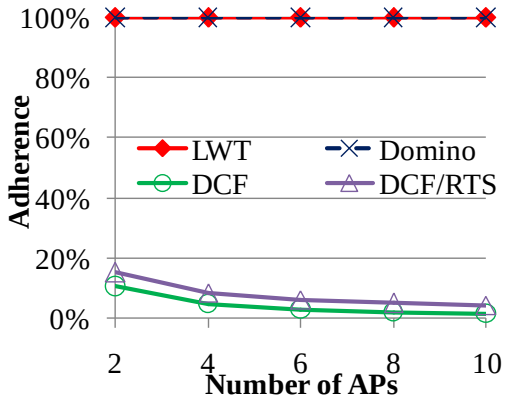


(b) Delay per packet

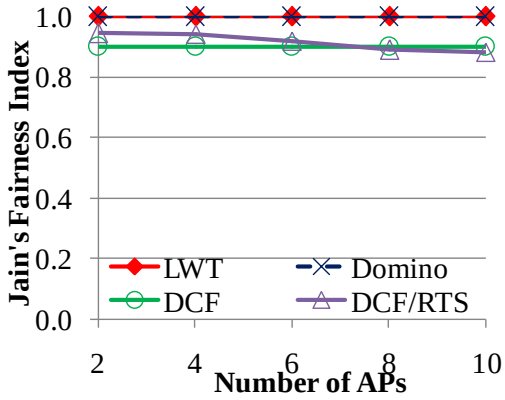
Figure 33: Performance comparison in fully connected topologies with saturated traffic (1/2)

Fig. 33(a) measures the throughput of LWT, DOMINO, DCF, and DCF/RTS. Due to transmitting according to a schedule and thus avoiding collisions, DOMINO and LWT achieve much higher throughput than DCF and DCF/RTS. DCF performs worse as the number of APs increases because of increasing contention. DCF/RTS performs worse than DCF in low contention scenarios since RTS/CTS generates large overhead, and performs better than DCF in high contention scenarios since RTS/CTS successfully decreases the overhead of collisions. Since DOMINO requires each AP to poll its clients for queue information periodically, the throughput of DOMINO slightly decreases as the number of AP increases.

Fig. 33(b) shows the average packet delay, which is the time between a packet's arrival



(a) Adherence to schedule

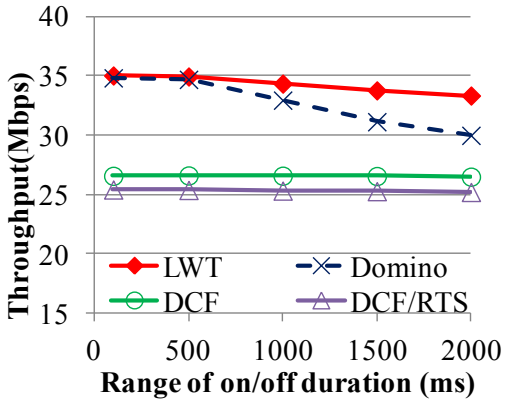


(b) Fairness

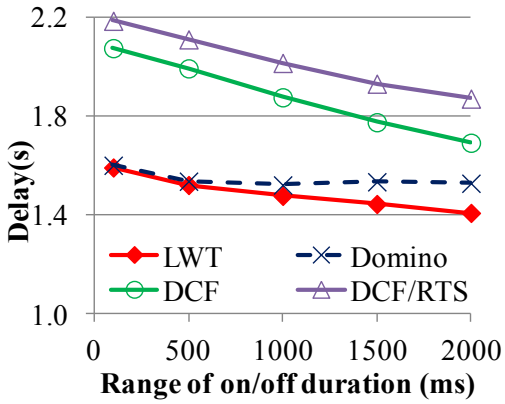
Figure 34: Performance comparison in fully connected topologies with saturated traffic (2/2)

at the queue and the time it is successfully transmitted. DCF and DCF/RTS have a higher delay due to larger contention time and more collisions. As the number of APs increases, delay increases due to longer schedule in LWT and DOMINO.

Fig. 34(a) presents the adherence of each mechanism. Since DCF and DCF/RTS does random backoff, their adherence is very low. LWT and DOMINO follow the schedule almost entirely. Fig. 34(b) measures fairness and all four mechanisms can provide real fairness in fully connected topologies.



(a) Network throughput



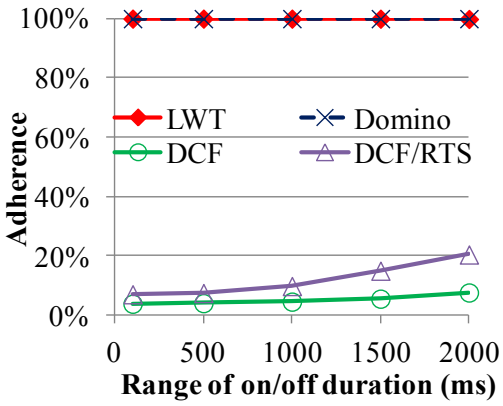
(b) Delay per packet

Figure 35: Performance comparison in fully connected topologies with dynamic traffic (1/2)

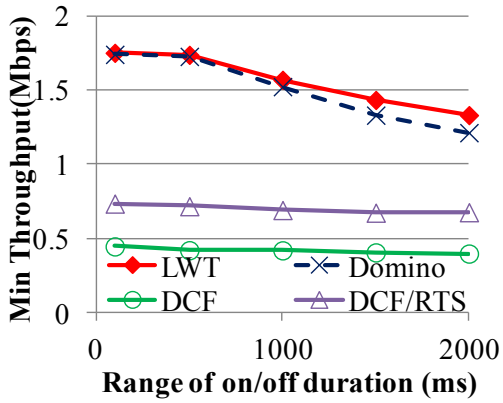
5.4.2.2 Non-backlogged nodes in fully connected topologies

In this section, we consider scenarios with non-backlogged nodes. We use dynamic on/off traffic in a fully connected topology with 2 APs, each with 5 STAs. We consider different traffic dynamic since there can be a significant diversity in traffic patterns/loads in practice. The on/off duration of traffic is randomly determined by each link during each on/off switch, and the range of randomness is changed from 100ms to 2000ms.

Fig. 35(a) shows the throughput of each mechanism under a different range of traffic on/off duration. The randomness of traffic doesn't affect DCF and DCF/RTS. As the range of traffic on/off duration increases, the average number of non-backlogged nodes increases. The throughput of LWT slightly decreases by cause of the increase in rand slots,



(a) Adherence to schedule

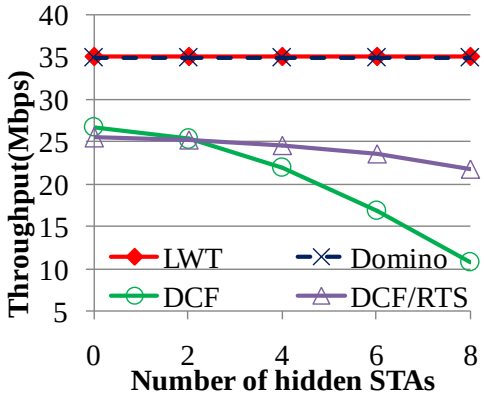


(b) Minimum throughput

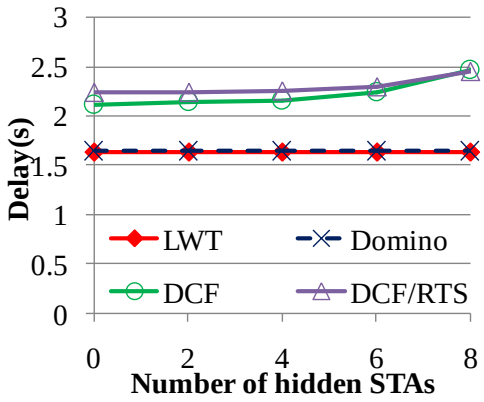
Figure 36: Performance comparison in fully connected topologies with dynamic traffic (2/2)

which have larger backoff time and collisions. The throughput of DOMINO drops due to missing schedules; that is, nodes cannot get scheduled quickly when it starts to get packets in the queue, and also can be scheduled while its traffic already enters the off period. This situation becomes more severe as the delay between the central controller and APs increases (the simulation setting of this delay is only 250 μs).

The packet delay of each mechanism is presented in Fig. 35(b). The delay decreases as the average number of non-backlogged nodes increases. Again, LWT and DOMINO performs better than DCF and DCF/RTS. Fig. 36(a) measures the adherence of different mechanisms. Note that when calculating adherence, rand slots are always considered as slots that successfully follow the schedule if the current scheduled link does not have a



(a) Network throughput



(b) Delay per packet

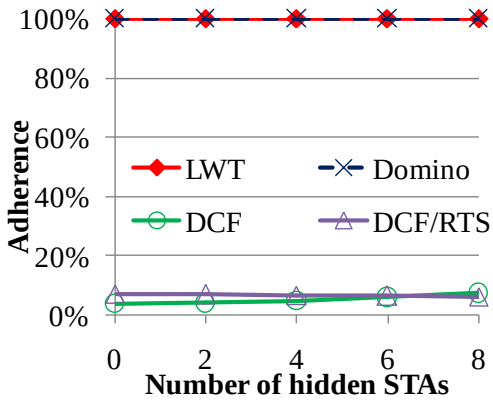
Figure 37: Performance comparison in topologies with hidden terminals (HtRx) (1/2)

packet to transmit. Again, DCF and DCF/RTS have very low adherence whereas LWT and DOMINO follow the schedule strictly.

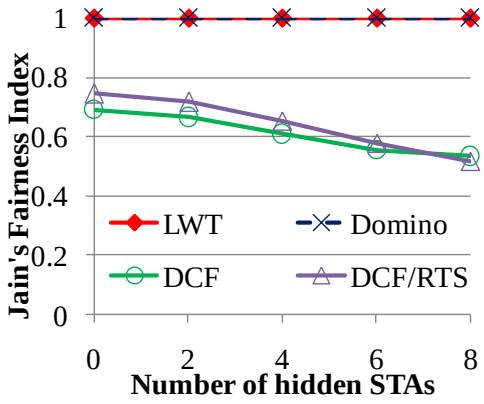
Instead of showing fairness, which is not very meaningful in dynamic traffic (each link has different traffic load), the minimum throughput among all links is presented in Figure 36(b). LWT gives the highest minimum throughput, and there is no starvation in any of the mechanisms.

5.4.2.3 Hidden terminals in a single collision domain

Although the probability of occurrence is not high, it is important to deal with hidden terminals. In CENTAUR [66], the observed ratio of hidden terminals is around 33% to 36%. More importantly, once hidden terminal problems happen, the performance degradation



(a) Adherence to schedule

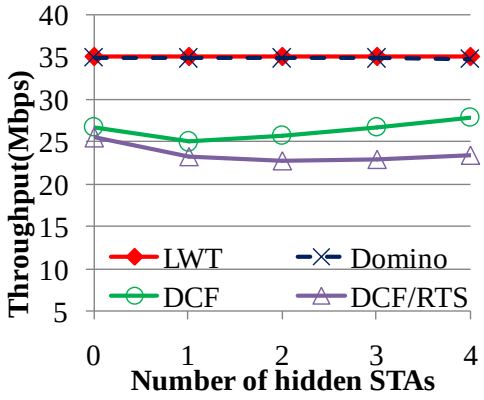


(b) Fairness

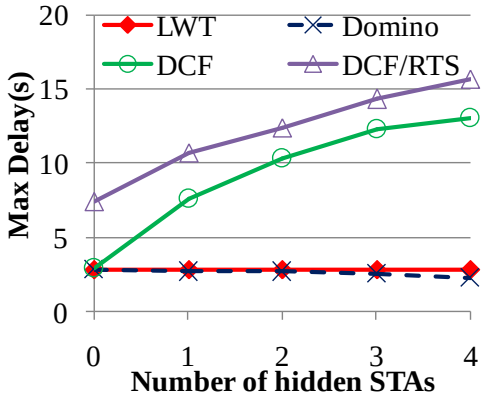
Figure 38: Performance comparison in topologies with hidden terminals (HtRx)(2/2)

brought by it can be tremendous. In Jigsaw [26], it was proposed that "co-channel interference from hidden terminals is likely the dominate cause of interference." Thus, we analyze the performance of different mechanisms in the presence of hidden terminals (HT). We set up two types of scenarios, HtRx (Fig. 29(b)) and HtNRx (Fig. 29(c)). All the topologies contain two APs; each has five STAs.

- *Hiddern Terminal (HtRx)*: Fig. 37(a), 37(b), 38(a), and 38(b) present the throughput, packet delay, adherence, and fairness of each mechanism in HtRx scenarios. Both LWT and DOMINO are not affected by hidden terminals. As the number of hidden terminals increases, throughput of DCF dramatically decreases due to collisions, which also increases the packet delay. Throughput of DCF/RTS performs better than



(a) Network throughput



(b) Max delay per packet

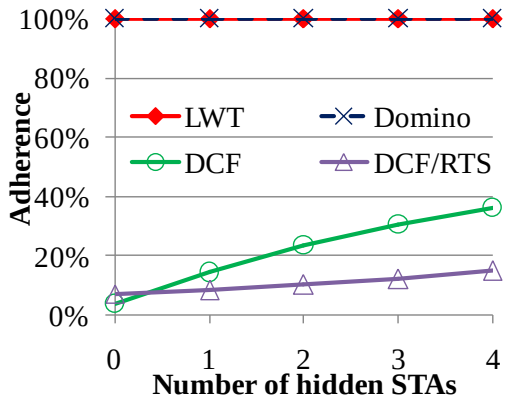
Figure 39: Performance comparison in topologies with hiddern terminals (HtNRx) (1/2)

DCF, but it still slightly decreases when the number of hidden terminals increases. With a large number of hidden terminals, DCF and DCF/RTS cannot achieve real fairness.

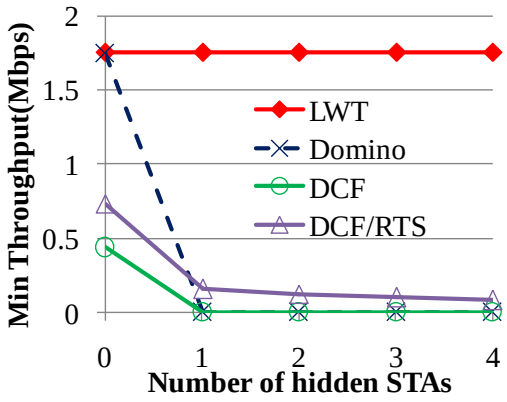
- *Hiddern Terminal (HtNRx)*: Fig. 39(a), 39(b), 40(a), and 40(b) present the throughput, maximum packet delay, adherence, and minimum throughput among all links of each mechanism in HtNRx scenarios. LWT is not affected by hidden terminals due to implementation of transparent transmissions.

  As indicated in Fig. 40(b), DOMINO, DCF, and DCF/RTS can have starvation under HtNRx scenarios.

  In DOMINO, the transmission of a link is triggered by the previous scheduled links.



(a) Adherence to schedule



(b) Minimum throughput

Figure 40: Performance comparison in topologies with hiddern terminals (HtNRx) (2/2)

A link can only be triggered if it can hear the Rx or Tx of the triggering link. If a scheduled link cannot be triggered by any of the previous scheduled links, it will be removed from the schedule and wait to be rescheduled by the central controller. In HtNRx, some nodes cannot hear the Rx and Tx of certain links. These nodes have fewer chances to be triggered by previous scheduled links, and thus get less chance to transmit. For example, in Fig. 29(c), link (D, C) can only be triggered by link (C, D). If the link scheduled before link (D, C) is link (A, B) or link (B, A), link (D, C) will not be scheduled. This causes starvation of link (D, C).

As illustrated in Section 5.3.4.1, DCF and DCF/RTS have unfairness problems under HtNRx scenarios since the transmission of certain nodes can be easily corrupted

by some other nodes. For example, in Fig. 29(c), transmission of node D can be corrupted by transmission of node A or B.

CHAPTER VI

SWITCH: DEDICATED SWITCHING MECHANISMS FOR FUTURE-PROOFING NETWORKS

6.1 Introduction

Amongst the myriad of approaches that have been considered for WiFi performance improvement, there has emerged an interesting technique to enable WiFi networks with a *lightweight control plane* without requiring any additional spectrum. One example of such a method is $Flashback$ [27]. Briefly, Flashback allows nodes in a WiFi network to send short control messages on the same channel concurrently with data transmissions, without harming the ongoing transmissions. These control signals can be thought of to represent a generic control plane that can be leveraged for a wide variety of applications. The central insight that allows for techniques such as Flashback to work is that transmissions typically operate with a non-zero *link margin*, and if the control messages are kept short enough, the damage they cause can be fixed using the available link margin. There are several other examples of similar techniques [43, 71], although we use Flashback as our core building block in this thesis.

Lightweight control creates a control plane without any additional requirements or severe negative impact. However, it has one significant constraint: *the transmitter and receiver of a transmission do not participate in the control plane for the duration of the transmission in any fashion that would impact that ongoing transmission.* The additional control plane capacity that exists due to the link margin is left for nodes other than the transmitter and receiver to exploit.

This attribute forms the context for this Chapter. We ask the following questions:

i) What benefits can be derived by allowing a transmitter and receiver of an ongoing

transmission also to participate in the seamless control plane?

ii) If benefits do exist, what techniques could allow the transmitter and receiver also to take part in the seamless control plane without compromising the ongoing transmission?

This proposed change is indeed significant because of the following reason: the transmitter and the receiver in tandem form the primary seat of intelligence for the ongoing data transmission. Hence, there are several decisions that they make that are not only significant but are decisions that only they can make, or only they have the information for making. Hence, allowing for the transmitter and receiver to exploit the seamless control plane during the ongoing transmission can allow for such decisions to be made better. In line with the terminology used in related works, we refer to such a control plane as an *inclusive seamless control plane*. We hasten to add that this argument for an $inclusive$ seamless control plane is not an argument for negating the contributions of techniques such as Flashback. However, the argument is in support for such participation to enable newer applications of the control plane, and hence even better performance.

In the previous Chapter, the transparent transmission in LWT demonstrates the benefits of *inclusive seamless control plane*. In this Chapter, we present several use-cases for an $inclusive$ seamless control plane. Briefly, the use-cases are as follows:

i) We identify topologies where having an inclusive control plane can solve an otherwise unaddressed hidden terminal problem.

ii) We show that packets that experience an irrecoverable collision (unrelated to the control plane) can be terminated immediately (like in protocols like CSMA/CD) and hence resources conserved by making the transmitter and the receiver participate appropriately in the seamless inclusive control plane.

iii) We consider the problem of backoffs in frequency domain [60]. We show that

through the use of an $inclusive$ seamless control plane, collision rates can be significantly reduced without the need for multiple radios. It can deal with the unfairness problem in hidden terminal topologies.

Note that the use-cases above are not meant to be new and novel in their conceptualization. Rather, our intent is to show that such use-cases are solvable and the solutions achievable in a much simpler fashion with less onerous requirements when an $inclusive$ seamless control plane is available.

We explore what it would take to facilitate such an $inclusive$ control plane. We argue that the fundamental requirement is for the transmitter and receiver to *switch* their mode of operation mid-stream from the ongoing data transmission/reception, use the seamless control plane, and then revert back to the data transmission/reception, and for all of this to be done without impacting the data. We show that such a *switch* mechanism is indeed feasible given state-of-the-art hardware and the nature of the lightweight control plane. We refer to our solution as simply $Switch$. We use experimental results from a WARP-based test-bed to both demonstrate and highlight properties of $Switch$. Using $Switch$ as the core primitive, we then propose algorithms for the three use-cases. ns-3 simulations are used to show the performance of $Switch$.

6.2 Background

6.2.1 Overview of Flash Signals

$Flashback$ [27] provides WiFi networks with a *lightweight control plane* without requiring any additional spectrum. Flashback allows nodes in a WiFi network to send short control messages on the same channel concurrently with data transmissions, without harming the ongoing transmissions.

The control messages are composed of several Flash signals. Flash signals are sinusoids with a frequency equal to a particular subcarrier of the current WiFi channel, and with a duration equal to an OFDM symbol (4μs). Since the Fourier transform of a sinusoidal wave

is a delta function, as shown through the experiments with Flashback, flash signals can be detected easily by simple peak detection algorithm without extra hardware requirements (the transmit/detect of flash signals can be achieved through software/firmware changes by reusing OFDM DSP blocks.). A similar technique has also been utilized in Back2F [60], where the detection rate is reasonably robust even with high self-interference.

Since WiFi networks have a fixed number of modulation types, transmissions typically operate with a non-zero link margin. That is, small damages in certain OFDM symbols on particular subcarriers won't affect the correct reception of a packet. With the redundancy provided by channel coding, the received packet still can be correctly decoded. The experiments of Flashback shows that packet receptions won't be disturbed by flash signals as long as the flash rate is less than 50,000 flashes per second [27].

6.2.2 Other Lightweight Control Plane Techniques

There are other related works and other special signals that also provide lightweight control. Hitchhike [43] provides a technique that utilizes the preamble field of a packet to carry control messages. Side Channel [71] uses the chip error pattern caused by interference to convey control messages. μACK [76] utilizes a portion of the current channel resources for control messages. Unique signals such as Gold codes [35] and correlatable symbol sequences (CSS) can be easily identified using correlation based detection. These special signals are used to compose effective control messages in different MAC protocols [78, 51]. The contributions in our research centered on the *Switch* mechanism could be extended to work with the above techniques and signals. However, we use Flash signals as our core building block due to its simplicity.

6.3 Switch

Although Flashback (or any of the other techniques [43, 71]) provides an extra lightweight control plane, the control plane is actively usable only by nodes other than the current transmitter and receiver. Since the WiFi radios are half-duplex, the transmitter cannot

Figure 41: Frame structure of WiFi

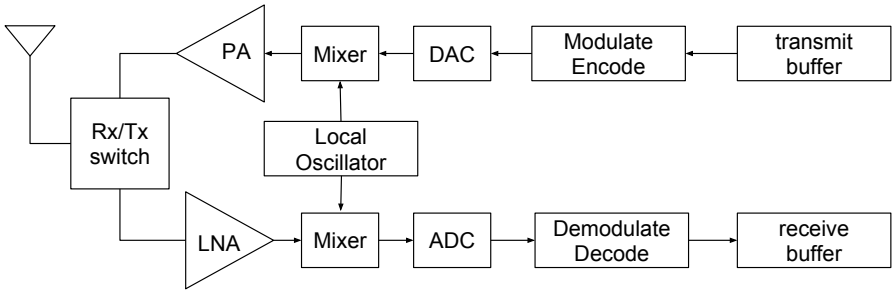


Figure 42: Simplified block diagram of an RF transceiver

receive, process, or propagate control information during transmission, and the receiver cannot transmit or spread control information during a reception. Thus, the transmitter and receiver of the ongoing transmission cannot participate in the control plane for the duration of the transmission in any fashion that would impact the current transmission. While we defer the discussion on how such participation might be useful to the next section, in this section we present a fundamental building block for supporting an inclusive control plane. The building block we offer is *Switch*, a mechanism that facilitates nodes to switch their communication modes (Tx to Rx and Rx to Tx) midstream of an ongoing transmission. We will explore how *Switch* can be used in different use cases in Section 6.4.

6.3.1 WiFi MAC and PHY Tx/Rx Basics

Figure 42 shows the basic design of a transceiver. Figure 43 shows the structure of the MAC and PHY Layer of WiFi [20]. Figure 41 shows the structure of a WiFi frame. The first portion of the PHY header is the PLCP preamble, which is used to identify the reception of a WiFi packet and for synchronization. When receiving a packet, the PHY layer

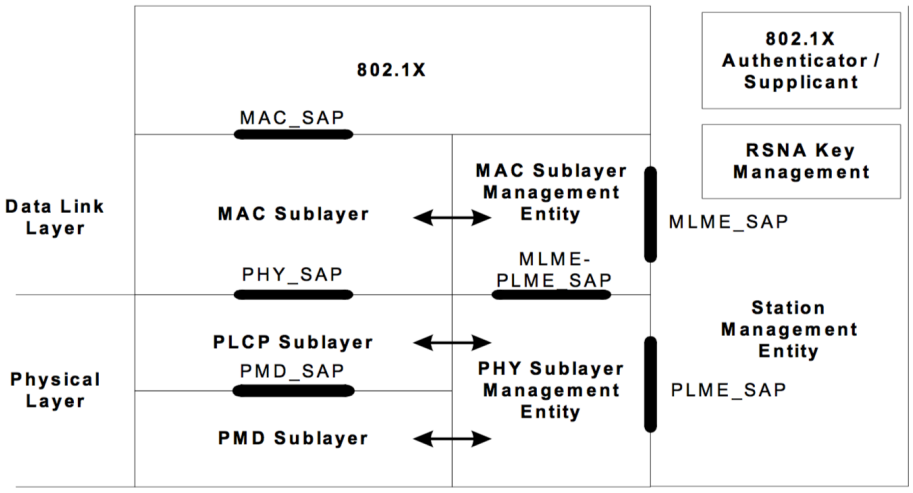


Figure 43: Architecture of the PHY and MAC Layer of WiFi

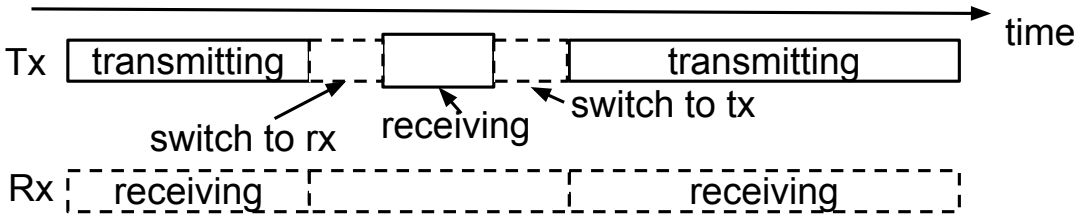


Figure 44: Timeline of Tx_Rx_Tx

identifies the PLCP preamble and starts to receive the signal. After appropriate signal processing (as shown in Figure 42) , the received bits are saved into the memory (receive buffer). If a Signal field (Figure 41) is successfully received, the PHY layer notifies the MAC layer of a packet arrival. The MAC layer can access the portion of the received packet that has been received and stored in the memory. Depending on the information indicated in the MAC header, such as the destination and the type of the received packet, the MAC layer waits until the completion of the reception and starts a proper response. The response can be sending ACK or CTS frames, stopping a timeout for ACK/CTS, or just ignoring the packet. When sending a packet, the MAC layer puts the packet into the memory (transmit buffer), starts the random backoff process, and indicates to the PHY layer to transmit the packet after the backoff timer expires. The MAC layer can keep monitoring

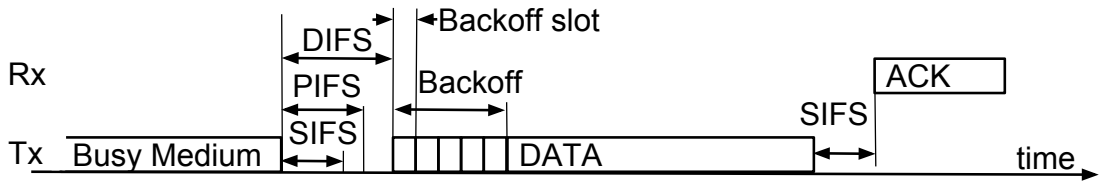


Figure 45: Timeline of DCF transmission

the transmission status (through reading the status registers), wait until the completion of the transmission, and start a proper response, which can be receiving ACK/CTS frames or starting a retransmission, etc.).

6.3.2 Switching Between Transmit and Receive

WiFi transmitters are half-duplex transmitters. They can switch between transmit and receive, but they cannot send and receive at the same time. The IEEE standard stipulates that the receive/transmit (Rx/Tx) turnaround time (aRxTxTurnaroundTime) of OFDM and high throughput (HT) PHY should be smaller than 2 microseconds [20]. State-of-the-art hardware typically has a Rx/Tx turnaround time of approximately 1 microsecond (e.g. transceiver MAX2829 [4]). Figure 45 shows the transmission timeline of WiFi. A wifi transmitter needs to switch to receive/transmit within SIFS duration after transmission/reception. SIFS is 10 microseconds and accommodates the PHY layer processing delay, MAC layer processing delay, and the receive/transmit turnaround time. Since the receive/transmit turnaround time is smaller than 2 microseconds, a transmitter actually can switch between transmit and receive much faster if there is no PHY/MAC processing delay. That is, if the decision to switch, the signal to be transmitted, and the timing of the switch are already determined before the switching, the transmitter can quickly switch between transmit and receive.

We propose two operation modes for *Switch* depending on whether the switch from Tx to Rx or from Rx to Tx.

- Tx_Rx_Tx: Figure 44 shows the timeline of a wifi transmitter switching to receive

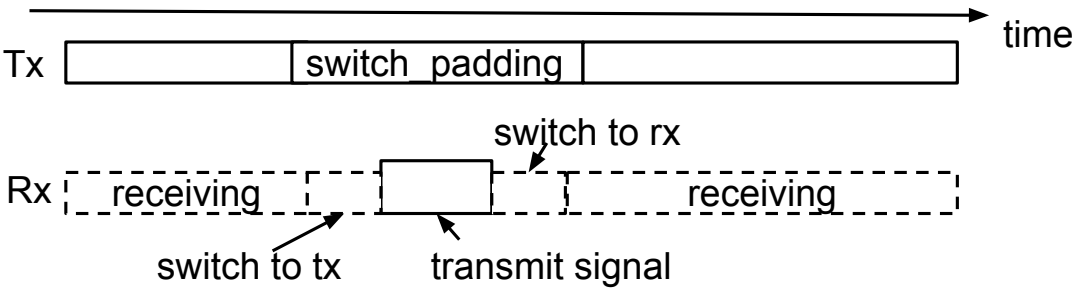


Figure 46: Timeline of Rx_Tx_Rx

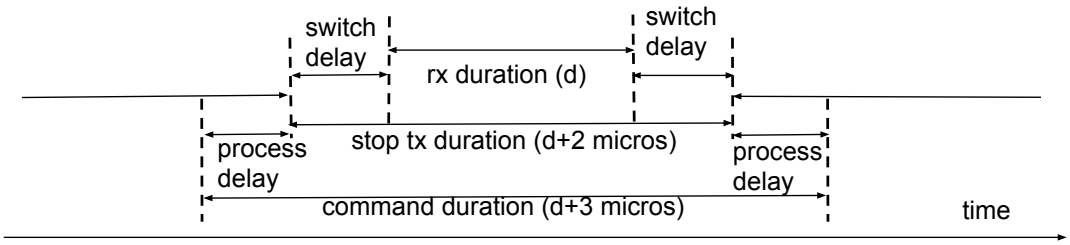


Figure 47: Duration of each section of Tx_Rx_Tx

during a transmission. During the transmission, the MAC layer starts a timer at the beginning of the transmission. The transmitter switches to receive when the timer expires. Another timer starts at the beginning of the switch, and the transmitter switches back to continuing transmission when the second timer expires. The "blank" portion during the reception can be easily filtered out by the receiver.

- Rx_Tx_Rx: Figure 46 shows the timeline of a wifi receiver switching to transmit during a reception. The MAC layer prepares the signal for transmission and stores it in a particular transmit buffer. During the reception, similarly, the MAC layer uses timers to control the switch. During the transmission period, the received signal will be garbage. Thus, the transmitter inserts a padding field, called *switch_padding*, in the transmitted packet when the receiver switches to transmit. The padding field *switch_padding*, is long enough to include the switching delay, and ensures that the whole packet can be received as if there is no switching happening at the receiver.

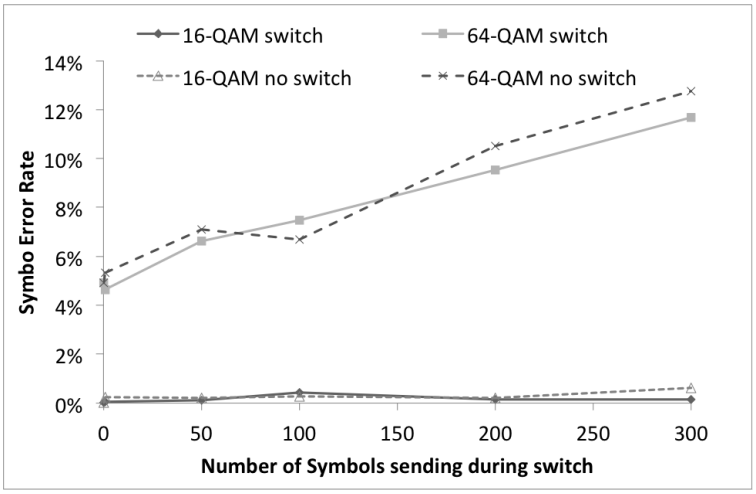


Figure 48: Symbol error rate of *Switch* (Tx_Rx_Tx)

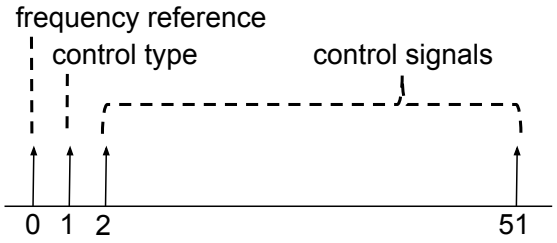


Figure 49: Structure of control signal

6.3.3 Some Properties of Switch

We conduct experiments that use WARP [15] radios to: (a) demonstrate that successful switching is indeed feasible; and (b) study detailed properties of *Switch* (e.g., how short or long the switch duration can be).

- Is *Switch* feasible?

  We implement both TX_Rx_Tx and Rx_Tx_Rx modes of *Switch*. When operating Tx_Rx_Tx mode, the experimental results confirm that: (a) the transmitter node can successfully receive signals during the switching period, and (b) the receiver node can remove the "blank" portion when the transmitter switches to receive, and decode the packet as if there were no switching happening at the transmitter. When operating Rx_Tx_Rx mode, the experimental results confirm that: (a) the receiver node can

successfully transmit signals during the switching period, and (b) the receiver node can decode the received packet under the protection of *switch_padding* as if there were no switching happening at the receiver.

- How short can switch duration be?

  Figure 47 shows the timeline of *Switch* operating Tx_Rx_Tx (the timing of Rx_Tx_Rx situation is similar to Tx_Rx_Tx, and thus is omitted.). There are command processing and switching delays when the MAC layer issues a switch operation. We set up two WARP nodes to measure the delays. First, one WARP node operates Tx_Rx_Tx while another WARP node receives the signal and measures the time duration of transmission stoppage during the switch. Then, one WARP node performs Tx_Rx_Tx while another WARP node transmits signals, and the first WARP nodes measure the time duration of receiving during the switch. Due to the command processing delay, the minimum receive period is 2 microseconds when the MAC layer issues a back-to-back switch command. The minimum duration of transmission stoppage is 4 microseconds, and the lowest delay for the entire processing of the command is 5 microseconds.

- How long can switch duration be?

  If the length of *switch_padding* is significant, the transmitted frame will also become large, and this can increase the packet error rate and symbol error rate due to increased frame size and loss of synchronization. We carry out experiments with two WARP nodes to study the effect of having a long duration switch (with a big *switch_padding*). One WARP node operates in Tx_Rx_Tx mode while the other WARP node receives the signal. Figure 48 shows the OFDM symbol error rate of the last 100 symbols of the received signal with different modulations. We compare the symbol error rate of operating with *Switch* and without *Switch* (but both have the

same frame size). Since the symbol error rate increases at the same speed, we conclude that the increased error rate is mainly due to the increased frame size (loss of synchronization), and the switching operation will not affect the reception afterward.

- How many times can *Switch* occur in a frame?

  As illustrated in the previous paragraph, *Switch* can happen as many times as required, as long as the resulting frame size is reasonable. Also, due to the switching and processing delays are shown in Figure 47, two switches can be scheduled back to back with the minimum separation period being the minimum command processing duration (5 microseconds).

- Does modulation affect *Switch*?

  Switch deals with the transmit and receive operation of RF signals. Thus, it is not directly impacted by different modulations. However, since *Switch* increases the frame size, as indicated in Figure 8, when the switch duration increases, the symbol error rate rises faster when using higher rate modulations (e.g. 64-QAM) compared to lower rate modulations (e.g. 16-QAM).

- Can off-the-shelf radios do *Switch*?

  Since switching between transmit mode and receive mode is a basic operation of a transceiver, as long as the software and firmware of the off-the-shelf radio can be changed, *Switch* can be supported by off-the-shelf radios without additional hardware requirements or changes.

6.3.4 Structure of Control Signal

Figure 49 shows the structure of control signal we use (Note that there are multiple ways to define this structure. We simply use a reasonable structure for our use-cases.). There are 52 subcarriers in 802.11 g and 56 subcarriers in the 20MHz channel in 802.11n/ac. We consider the case with only 52 subcarriers. The first subcarrier is used as a reference

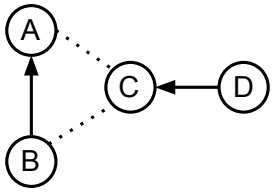


Figure 50: Topology with one-way hidden terminals

frequency so the frequency offset won't affect the identification of subcarriers. The second subcarrier is used to indicate the type of the control signal, and the rest of the subcarriers can indicate 50 different values (each subcarrier indicates a unique value). We will illustrate the detailed meaning of each control signal in the following use-case studies.

6.4 Use cases

In this section, we use *Switch* as the key building block for solving three different problems in WiFi networks. Note that the problems themselves are certainly solvable through means other than *Switch*. The contributions lie in showing that *Switch* can solve these problems in an elegant and efficient fashion.

6.4.1 Extending the range of Carrier Sense

6.4.1.1 Starvation in One-way Hidden Terminal

We propose an algorithm based on *Switch* to deal with the starvation problem in one-way hidden terminal scenarios. Figure 50 shows an example of a one-way hidden terminal: Node D cannot hear node A and node B, but when either node A and node B transmits, a transmission by node D will experience a collision at node C. In this situation, it is challenging for node D to get a chance to send to node C. This results in a starvation problem.

6.4.1.2 State-of-the-art Solutions

The standard solution to solve this starvation problem is exchanging RTS/CTS packets before every data transmission. However, since the RTS/CTS transmit duration can be

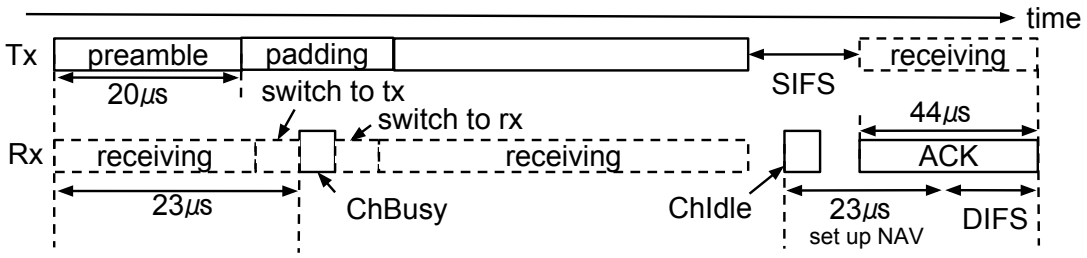


Figure 51: Timeline of ECS transmission

long (52 μs and 44 μs when using 6Mbps), and the channel can only be reserved after the successful transmission of CTS, RTS/CTS cannot effectively solve the starvation problem (as will be seen in the evaluation section). Also, it is well established that the exchange of RTS/CTS before every data transmission introduces large overheads. RTS/CTS transmissions count for 26% of a frames total transmission time when transmitting 1500 byte packets at 54Mbps. Related works Centaur [66] and 802.11ec [51] can deal with the starvation problem. Centaur schedules the transmissions of hidden terminals far from each other to avoid collisions. However, Centaur requires a central controller, which is not always necessarily present in all network environments. 802.11ec decreases the time for channel reservation by using correlatable symbol sequences (CSS) to replace RTS/CTS. However, the usage of CSS introduces a second addressing system, which increases the complexity. Also, as the number of nodes increases, the code length of CSS needs to increase to support more addresses for each node, and this will increase the channel reservation time.

6.4.1.3 Switch-based Solution

We utilize *Switch* to propose a new algorithm, Extended Carrier Sense (ECS), to decrease the channel reservation time with a small control overhead. Figure 51 shows the timeline of a transmission of ECS. When nodes hear channel busy, after receiving the preamble, they switch to transmit a flash signal, *ChBusy*. This includes the receiver of the ongoing transmission. Also, at the end of the reception, during the SIFS duration, nodes again switch to transmit another flash signal, *ChIdle*. Nodes other than the receiver do not need

to send an ACK and hence switch back to receive. The receiver goes on to transmit the ACK. When any node hears a *ChBusy*, it sets up a particular long backoff time (set to the maximum packet transmit duration), ECSbackoff, to defer the channel access. When the node receives *ChIdle*, it stops the ECSbackoff and sets up NAV (23 μs) to avoid collision with the ACK.

Since channel busy and channel idle are both very simple information that does not require large processing delay, it is possible to use *Switch* as the underlying mechanism and achieve small channel reservation time. *What is critical for this mechanism to be effective is for the receiver to switch midstream of its reception to transmit ChBusy and ChIdle, something enabled by switch.* The total switch duration takes 8 μs, and we use a *switch_padding* of 12 μs to protect the data reception. Since every node receives the packet switches and transmits ChBusy, nodes in the same collision domain learn that the channel has become busy, and the channel is effectively reserved. The channel reservation time of ECS is 23 μs, which is 37% of the channel reservation time (62 μs) when using RTS/CTS. The control overhead introduced by ECS is the 12 μs *switch_padding*, which is 10% of the control overhead (116 μs) when using RTS/CTS.

When there are multiple collision domains, each collision domain can have its own *ChBusy* and *ChIdle* signals and operate orthogonally. The 54 subcarriers in a 20MHz channel can support 25 different collision domains (25 signals for *ChBusy* and 25 signals for *ChIdle*) assuming a single subcarrier is used as the "address" of a domain. Addresses can be reused for collision domains that are not next to each other. Algorithm 12 summarizes the core mechanism of ECS using pseudocode.

6.4.2 Early Collision Termination (Micro CTS without RTS)

6.4.2.1 Avoiding Collisions

Since DCF is a contention based multiple access control, in WiFi networks, collisions can occur frequently and decrease the system throughput. The situation becomes severe (as will

Algorithm 12 Switch-ECS

```
1: if Receive Preamble then
2:     switch to transmit ChBusy
3:     switch back to receive
4:     wait until channel idle and switch to transmit ChIdle
5: else if Receive ChBusy of its own collision domain then
6:     defer and set up a long backoff ECSbackoff
7: else if Receive ChIdle of its own collision domain then
8:     stop ECSbackoff and setup NAV to protect ACK
9: end if
```

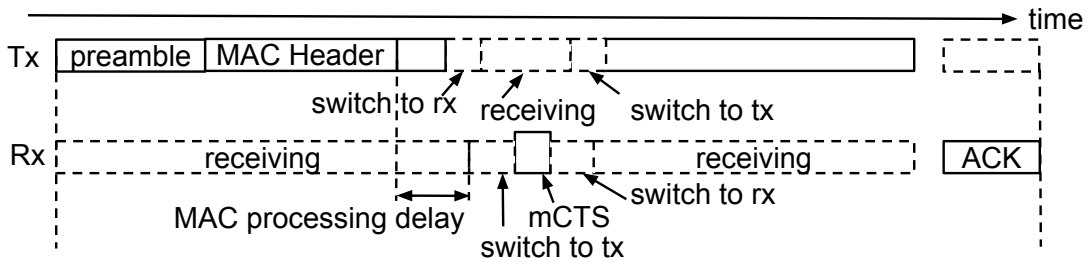


Figure 52: Timeline of mCTS transmission

be shown in the evaluation section) when there are hidden terminals, or when the number of contending nodes is large.

6.4.2.2 State-of-the-art Solutions

RTS/CTS control packet exchange is the standard technique used to prevent collisions from occurring in the first place. However, as mentioned in the previous section, RTS/CTS incur significant control overheads. CSMA/CN [61] is a technique proposed to decrease the overheads of collisions without extra control costs. In CSMA/CN, the receiver monitors the receiving status and notifies the transmitter to stop transmission once a collision is detected. Though useful, CSMA/CN requires two antennas for operation. Like 802.11ec, CSMA/CN also requires usage of unique signal signatures for additional addressing, which increases the complexity, and the overheads increase as the number of nodes increases.

6.4.2.3 Switch-based Solution

We use *Switch* to propose an algorithm, Micro CTS (mCTS), to decrease the collision overhead while not incurring any significant control overheads. Figure 52 shows the timeline

of a transmission of mCTS. The transmitter transmits the data frame till the MAC header and continues to send for a small duration (8 μs) to accommodate the MAC processing delay and the switch to receive delay at the receiver. The receiver receives the data frame till the MAC header, checks to see if the destination address matches its MAC address. If the destination address matches its MAC address, and the MAC header is being correctly received, the receiver switches to transmit a flash signal *fmCTS* at a predetermined duration (10 μs) following receipt of the MAC header. The receiver checks the confidence of the received bits via physical-layer hints from SoftPHY [42, 70] to make sure of the correctness of the reception of the MAC header. Meanwhile, the transmitter waits for the *fmCTS*. If it doesn't detect the *fmCTS*, the transmitter stops transmitting and goes into its retransmission sequence. If it detects the *fmCTS*, it switches back to continue sending. The total switching duration of the receiver takes 8 μs, and the transmitter uses 12 μs for its fmCTS timeout.

The mechanism of mCTS is quite similar to RTS/CTS, but with significantly smaller overheads. The first portion of the data frame transmitted by the transmitter acts implicitly as the RTS. Although this part is not as robust as an RTS (RTS uses the smallest modulation rate), we show in the evaluation section that because the control overheads are much lower, mCTS results in much better system throughput. *What is critical for this mechanism to be effective is for the receiver to switch midstream of its reception to transmit* fmCTS*; and the transmitter to switch midstream of its transmission to receive* fmCTS*, and switch back to continue the transmission.* The total overhead of mCTS is the time for receiving *fmCTS*, which is 12 μs, which is 10% of the control overhead (116 μs) of RTS/CTS.

Again, when there are multiple collision domains, each collision domain can have its own *fmCTS* recognizable signals and operate orthogonally (same as *ChBusy* and *ChIdle* in ECS). Note that it is sufficient for *fmCTS* to represent an individual collision domain rather than a particular transmitter. Since there can be only one successful transmission in a single collision domain, a receiver node will not reply with an *fmCTS* if there is more than one node transmitting in the same collision domain. Algorithm 13 summarizes the

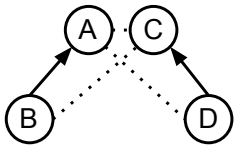


Figure 53: Topology with two-way hidden terminals

Algorithm 13 Switch-mCTS

```
1: Transmiter:
2: Transmit till a predefined time and switch to receive
3: Set up a fmCTS timeout timer
4: if Receive fmCTS then
5:     switch to transmit the reset of the packet
6: else if fmCTS timeout timer expires then
7:     start over and arrange a retransmission
8: end if
9: Receiver:
10: Receive till MAC header and check its correctness
11: if Correctly receive the MAC header, and the destination matches my_address then
12:     switch to transmit fmCTS at a predefined time
13:     switch back to receive
14: end if
```

core mechanism of mCTS using pseudocode.

Note that, unlike CTS, *fmCTS* does not have the ability to reserve the channel (it does not have an NAV field). Thus, mCTS will indeed have a problem when there are hidden terminals. However, in combination with the ECS explained earlier (referred as ECSmCTS), hidden terminals can be tackled as well. When operating ECSmCTS, nodes wait until *fmCTS* timeout to transmit *ChIdle*. The second subcarrier, which indicates the type of the control signal, can be used to identify *fmCTS* from *ChBusy* and *ChIdle*.

6.4.3 Improving WiFi Backoffs

6.4.3.1 Unfairness in Hidden Terminal Scenarios

As mentioned in Section 6.4.1, there are unfairness and even starvation problems in one-way hidden terminal topologies. The unfairness problem persists in two-way hidden terminal topologies as well. For example, in Figure 53, node B and D cannot hear each other, while node A and C can hear all the nodes. Since node B and D cannot listen to the transmission of some nodes, they tend to have collisions. Thus, these nodes converge to using

larger contention windows and get fewer chances to transmit, which leads to the unfairness problem.

6.4.3.2 State-of-the-art Solutions

RTS/CTS can not eliminate the unfairness problem since the CTS of node B and node D are also tend to get collisions. Related works such as Rhythm [64], LWT [62], and Domino [78] can deal with this unfairness problem by appropriately scheduling transmissions. However, all these mechanisms require a centralized network structure with a controller.

6.4.3.3 Switch Solution

We use *Switch* to propose a novel algorithm, flash Backoff (fBK), to deal with the unfairness problem. The problem of time domain backoff is that the time delay of channel reservation (the time from start of RTS to the time when CTS completes) is also being counted as the backoff time. If the backoff time difference between two hidden terminals is smaller than the channel reservation delay, collisions happen. Thus, instead of doing backoff in the time domain, fBK carries out backoff using flash signals (frequency domain), and appropriately propagates the backoff information for hidden terminals. This clearly separates the delay of channel reservation and the backoff value, and thus gives better fairness. The idea of implementing backoff in the frequency domain rather than time domain is the same as Back2F [60]. However, Back2F requires two antennas for operation. Without any additional mechanisms, Back2F can trigger repetitive collisions when hidden terminals exist. Back2F also cannot deal with the unfairness problem in the hidden terminal topologies.

We first introduce how to use flash signals to perform backoffs (fBK). Then, we combine fBK with ECS (ECSfBK) to properly propagate the backoff and channel busy information.

fBK backoff without hidden terminals:

Figure 54 shows the timeline of backoff mechanism of fBK, in which nodes broadcast a

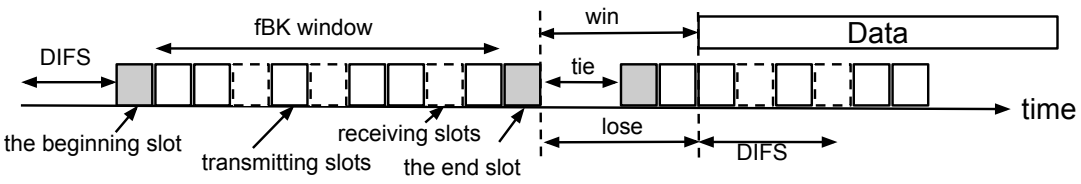


Figure 54: Timeline of fBK backoff mechanism

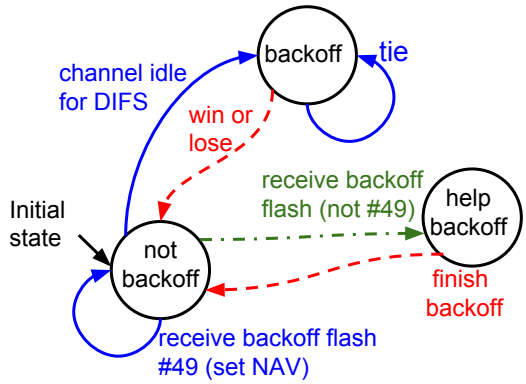


Figure 55: State Diagram of ECSfBK

randomly selected flash signal, and randomly switch to observe the flash signals transmitted by others. fBK backoff mechanism contains a fixed size of fbk backoff slots. The first slot is called the beginning slot, and the last slot is the end slot, and the rest of the slots are either a receiving slot or a transmitting slot. Before each backoff, nodes randomly select a backoff number f_m from [0,47]. The value of f_m determines the flash signal being transmitted during the backoff duration.(f_m=0 matches to the third subcarrier, f_m=1 matches to the fourth subcarrier, etc.) Nodes also randomly select k different numbers (rx_is) from [1, fbkWindow], where fbkWindow is the fBK window size, which is the number of fbk backoff slots between the beginning slot and the end slot ($k = 3$ and fbkWindow=9 in Figure 54). rx_is are the indices of the receiving slots, and the other slots are transmitting slots.

The key property of fBK backoff is the ability to efficiently learn the backoff value (f_m) selected by other nodes, which is enabled by effectively switching to receive during transmission. One node can learn if it: (i) wins the contention (having the smallest f_m), (ii) lose the contention (receive a flash signal less than its own f_m), or (iii) it is a tie. That is, it receives a flash signal having the same value of its f_m, and no other smaller flash signals

are being received. The learning of tie can mostly avoid collisions. If there is a tie, nodes having the same smallest f_m enter another round of contention. Since only nodes have the smallest f_m will this contention, the number of contending nodes decreases quickly, and the probability of having a tie decreases rapidly. While efficiently reducing the collision rate, knowledge of the exact backoff values also gives better fairness compared to time domain backoff, in which channel reservation delay biases the backoff value.

Algorithm 14 Switch-fBK

```
1:  function StartBackoff
2:      if f_m == 0 then
3:          random select f_m from [0,47]
4:      end if
5:      random select rx_i from [0,fbkWindow]
6:      Counter=-1
7:      FBackoff()
8:  end function
9:  function FBackoff
10:     Counter++
11:     if Counter==fbkWindow+2 then
12:         if (f_r > f_m and f_e == f_m) or f_e > f_m then
13:             set f_m = 0
14:             schedule transmit after d_win
15:         else if f_e == f_m then
16:             set f_m = 0
17:             schedule StartBackoff() after d_tie
18:         else
19:             set f_m = f_m - f_e
20:             schedule CCA after d_lose
21:         end if
22:         return
23:     else if Counter==fbkWindow+1 then
24:         if f_r >= f_m then
25:             switch to receive
26:         else
27:             transmit flash signal f_r
28:         end if
29:     else if Counter equals to any rx_i then
30:         switch to receive
31:     else
32:         transmit flash signal f_m
33:     end if
34:     FBackoff() (go to next backoff slot)
35: end function
```

We descript the detailed of fBK below. Nodes wait for DIFS after the channel becomes idle, and transmit the flash signal determined by f_m during the beginning slot. Then, nodes

will keep transmitting the flash signal during transmitting slots and switch to receive during the receiving slots. The receiving slots help a node to determine if it has selected the smallest f_m and hence won the contention. If there is a tie, nodes with the smallest f_m start another backoff procedure. The node that wins the contention initiates a transmission after a delay d_{win}. Nodes that are having a tie start another fbk backoff after a delay d_{tie} (note that only the nodes experiencing a tie will attend this fbk backoff). Nodes that lose the contention start a timer of d_{lose} and wait for channel idle for DIFS after the timer expires. The delay after fBK backoff determines the priority of behaviors. We select $d_{tie} < d_{win} = d_{lose}$. If the selected set of rx_is of some nodes are the same, some nodes will result in a tie while some nodes observe a win. In this situation, nodes resulting in a tie will start another fBK backoff before the nodes seeing a win initiate the transmission. Nodes finding a win will defer when they hear the flash signals transmitted during the beginning slot.

Since nodes cannot observe each other if they happen to select the same set of rx_i, the end slot is used to help nodes figure out the smallest f_m among all nodes. Nodes that observe win or tie will switch to receive during the end slot. Other nodes that already lose transmits the smallest flash signal they saw during the end slot.

Algorithm 14 summarize the core mechanism of fBK backoff using pseudocode. During the k receiving slots, nodes record the smallest subcarrier it receives, f_r. Nodes compare it with f_m. If $f_m > f_r$ (line 26), nodes will transmit f_r during the end slot; else if $f_m <= f_r$ (line 24), nodes receive during the end slot, and record the smallest subcarrier f_e. If $f_r > f_m$ and $f_e = f_m$, or if $f_e > f_m$ (line 12), the node wins the contention; else if $f_e == f_m$ (line 15), it is a tie, and the nodes will start another fbk backoff; else (line 18), nodes lose the contention.

Combining ECS and fBK backoff:

fBK backoff can work well and efficiently avoid collisions when there are no hidden terminals. However, since hidden terminals cannot detect the start of a repeated fBK backoff (due to a tie) or the start of a transmission, fBK backoff can trigger repetitive collisions when

Table 10: ns-3 parameter

Parameter	Value
Frame size	1500byte
Basic transmission rate	6Mbps
Data transmission rate	54Mbps
Slot time	9μs
SIFS	10μs
DIFS	28μs

hidden terminal exists. Thus, we combine ECS with fBK backoff and propose ECSfBK. Again, the second subcarrier, which indicates the type of the control signal, can be used to distinguish backoff flash signals from *ChBusy* and *ChIdle*.

Since ECS can indicate the start of a transmission, the only thing left is to indicate the start of a repeated fBK backoff successfully. Similar to the concept in ECS, nodes that do not attend an fBK backoff enters a "help backoff" state after receiving the first backoff flash signal during the beginning slot. After the beginning slot, nodes in "help backoff" state will transmit $f_m = 49$ (the largest subcarrier) during the first backoff slot. Other nodes hearing the backoff signal with $f_m = 49$ will set up NAV equal to the rest of the backoff duration. Nodes in "help backoff" state then receive through the rest of the backoff slots and transmit the observed f_r during the end slot. Figure 55 illustrates the state diagram of ECSfBK.

6.5 Evaluations

In this section, we carry out ns-3 simulations [10] to evaluate the performance of each algorithm based on *Switch*.

6.5.1 Simulation Setup and Methodology

We present the performance evaluation of ECS, ECSmCTS, ECSfBK, DCF, and DCF with RTS/CTS (DCF/RTS). Table 10 shows the simulation parameters, which follows 802.11g. In all scenarios, traffic is generated on all uplinks and downlinks, and nodes are always backlogged. We first evaluate the performance of each mechanism under the targeted topologies (one-way hidden terminals, two-way hidden terminals, and fully connected

topologies) with 6 to 30 nodes. Then, the performance of ECSmCTS and ECSfBK is evaluated under random topologies with 16 to 40 nodes. We use total system throughput to evaluate the efficiency of each protocol, and use minimum throughput per node to evaluate the fairness provided by each protocol.

6.5.2 ECS

We evaluate the performance of ECS in a topology with one-way hidden terminals (Figure 50). We set up 2 APs, each with five clients, and change the number of nodes that suffer the starvation problem (Node D in Figure 50). Figure 56(a) and 56(b) shows the system throughput and minimum throughput per node. While the system throughput of each algorithm is similar, the starvation problem of DCF is severe. DCF with RTS/CTS can slightly reduce the starvation problem, while ECS gives much better fairness. The minimum throughput per node of ECS is 6x to 12x of that of DCF with RTS/CTS when there is one-way hidden terminal.

6.5.3 ECSmCTS

We evaluate the performance of ECSmCTS in a fully connected topology and a topology with two-way hidden terminals (Figure 53)

In fully connected topologies, we set two clients to each AP, and increase the number of AP from 2 to 10 (the number of nodes increase from 6 to 30). Figure 57(a) and 57(b) shows the system throughput and minimum throughput per node with different number of nodes in a fully connected topology. As shown in Figure 57(b), all algorithms give similar fairness among nodes. As shown in Figure 57(a), when the number of nodes is small, the collision rate is low, and DCF with RTS/CTS performs worse than DCF due to significant control overhead. As the number of nodes increases, the collision rate increases, and thus the throughput of DCF becomes worse than DCF with RTS/CTS. ECSmCTS, avoiding collision overhead with small control cost, performs better than DCF and DCF with RTS/CTS

under every collision rate. The maximum throughput improvement of ECSmCTS compared to DCF and DCF with RTS/CTS are 22% and 17% respectively.

In two-way hidden terminal topologies, we set up 2 APs, each with five clients. We change the number of nodes suffers two-way hidden terminal (node B and node D in Figure 53). When the number of hidden terminals is zero, the topology is fully connected. Figure 58(a) and 58(b) shows the system throughput and minimum throughput per node with a different number of hidden terminals in two-way hidden terminal topologies. Due to the ECS mechanism, ECSmCTS gives much better minimum throughput compared to DCF and DCF with RTS/CTS. ECSmCTS also provides better system throughput due to avoiding collision overhead with small control cost. The average improvement of ECSmCTS compared to DCF and DCF with RTS/CTS are 55% and 17% respectively.

6.5.4 ECSfBK

We evaluate the performance of ECSfBK in fully connected topologies, the two types of hidden terminal topologies, and random topologies. We use $k = 3$, fbkWindow=9, $d_{tie} = 4\mu s$, and $d_{win} = d_{lose} = 14\mu s$. Each backoff slot is 5$\mu$s (it is the shortest time of switch as illustrated in Section 6.3.3). Nodes keep transmitting flash signals and switch to receives for 2μs during a receiving slot. We also present results for ECSmCTS as a comparison.

In fully connected topologies, we set two clients to each AP, and increase the number of AP from 2 to 10 (the number of nodes increase from 6 to 30). Figure 59(a) and 59(b) show that ECSfBK efficiently avoids collisions (having similar throughput as ECSmCTS) and give reasonable fairness in fully connected topologies.

In two-way hidden terminal topologies (Figure 53), we set up 2 APs, each with five clients, and change the number of nodes suffer two-way hidden terminal (node B and node D in Figure 53). When the number of hidden terminals is zero, the topology is fully connected. In one-way hidden terminals (Figure 50), we set up 2 APs, each with 5 clients, and change the number of nodes that suffer the starvation problem (Node D in Figure 50).

Figure 60(a), 60(b), 61(a), and 61(b) show that ECSfBK, which efficiently utilize *swtich* to carry out backoff in frequency domain in hidden terminal topologies, gives much better minimum throughput while maintaining good system throughput in both hidden terminal topologies.

Finally, we carry out random topologies. We set up 2 APs in middle of an area, and randomly place a different number of clients (16 to 40) around the 2 APs. Figure 62(a) and 62(b) show the performance of each algorithm in random topologies with a different number of clients. ECSfBK gives good system throughput and minimum throughput per node. The max/min/average improvement in throughput of ECSfBK compared to DCF and DCF with RTS/CTS are 8%/1%/4% and 12%/8%/10% respectively. The increase in minimum throughput per node of ECSfBK compared to DCF and DCF with RTS/CTS are 77%/16%/40% and 776%/97%/240% respectively.

6.6 Related Work

6.6.1 Solutions for Hidden Terminal Problems

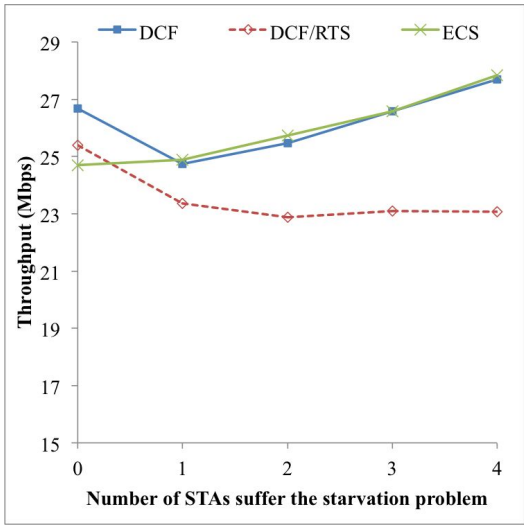
As mentioned in Section 6.4.1, 802.11ec [51] and Centaur [66] can deal with the starvation problem in one-way hidden terminal scenarios. However, Centaur requires a central controller, an entity not always present in all network environments, and 802.11ec requires usage of another addressing system for all the nodes. As mentioned in Section 6.4.3, Rhythm [64], LWT [62], and Domino [78] can deal with the unfairness problem in hidden terminal topologies. However, all these mechanisms require a centralized network structure with a central controller. Zigzag [36] can recover data from repetitive collisions that occur due to hidden terminals. However, since it requires at least two collisions from the same two frames, something that will not occur in one-way hidden terminal topologies, Zigzag cannot deal with the starvation problem in the one-way hidden terminal scenario.

6.6.2 Solutions for Avoidance of Collisions

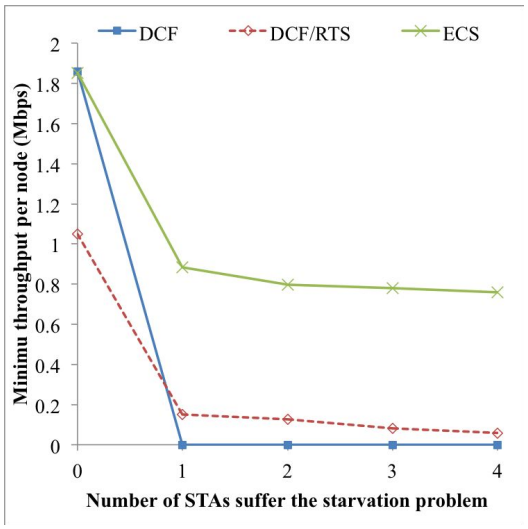
As mentioned in Section 6.4.2, CSMA/CN [61] is proposed to decrease the overhead of collisions without extra control costs. However, CSMA/CN requires two antennas for operation. It also requires usage of another addressing system for all the nodes. L-BEB [25] and CF-MAC [59] use determined backoff values to avoid collisions. However, these approaches rely on RTS/CTS to deal with hidden terminal problems, and this, in turn, introduces significant control overheads. Soft-TDMAC [29] avoids collisions by scheduling transmissions. However, it requires tight time synchronization for sufficient efficiency, which is an additional burden. Chain [75] reduces collisions by triggering transmissions using information from overheard transmissions. However, it cannot deal with collisions generated by hidden terminals.

6.6.3 Backoffs in the Frequency Domain

Back2F [60] is a mechanism that efficiently does backoffs in the frequency domain. Though Back2F achieves low collision rates with small backoff overheads, Back2F requires two antennas for operation. Also, Back2F does not deal with hidden terminal problems. FICA [68] also utilizes signals in the frequency domain for channel access. However, it requires RTS/CTS exchanges that incur significant control overheads.

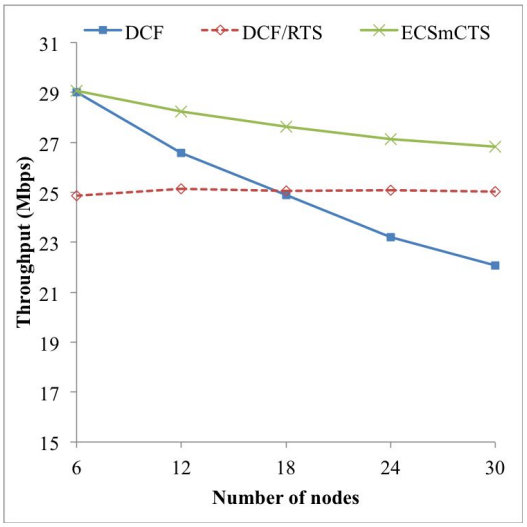


(a) Throughput

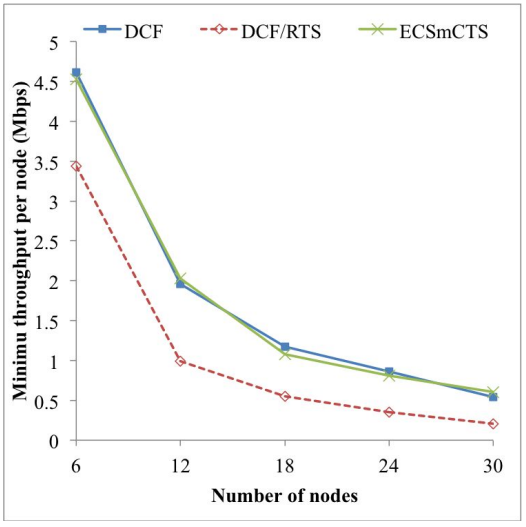


(b) Minimum throughput per node

Figure 56: Performance of ECS in topologies with one-way hidden terminals

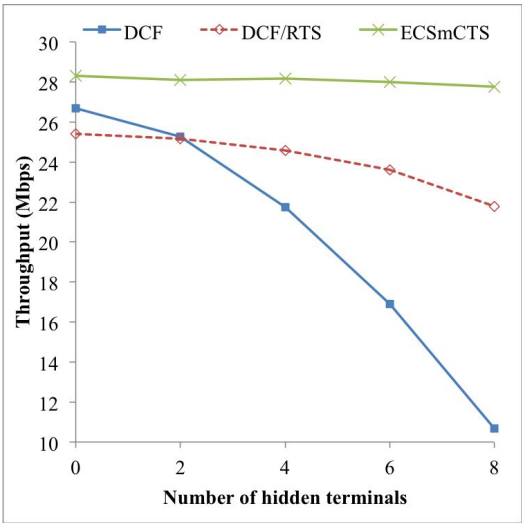


(a) Throughput

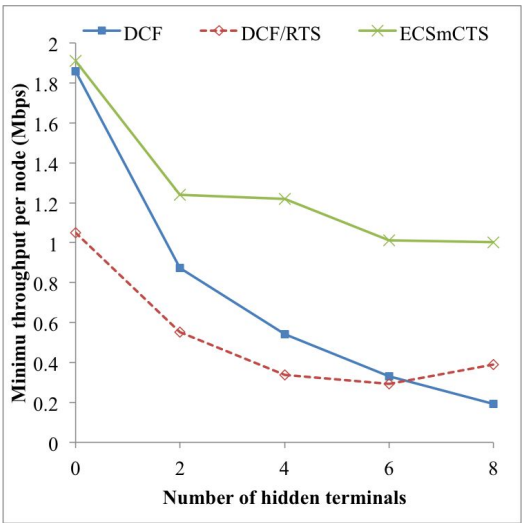


(b) Minimum throughput per node

Figure 57: Performance of ECSmCTS in fully connected topologies

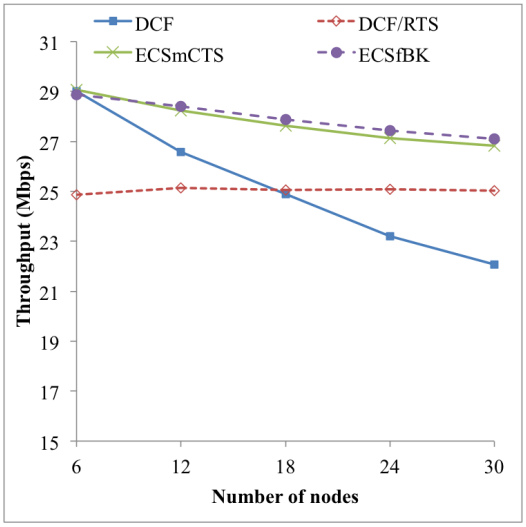


(a) Throughput

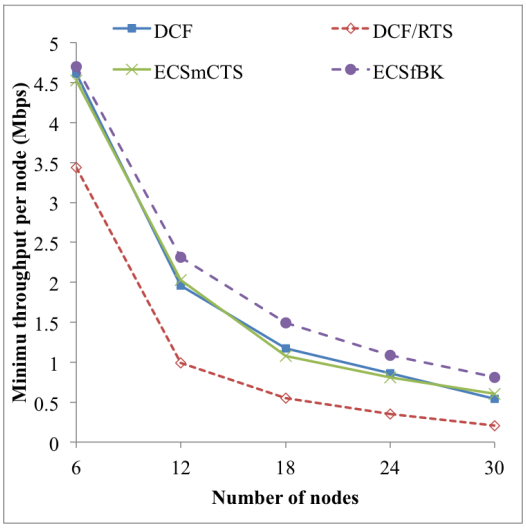


(b) Minimum throughput per node

Figure 58: Performance of ECSmCTS in topologies with two-way hidden terminals

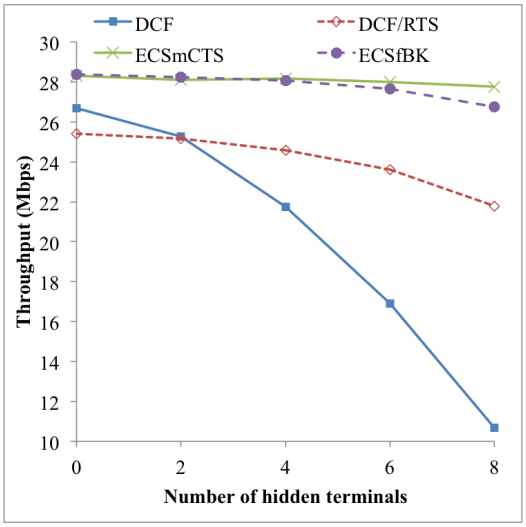


(a) Throughput

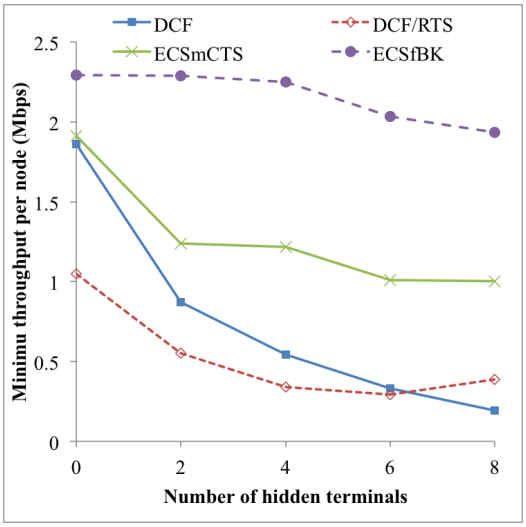


(b) Minimum throughput per node

Figure 59: Performance of ECSfBK in fully connected topologies

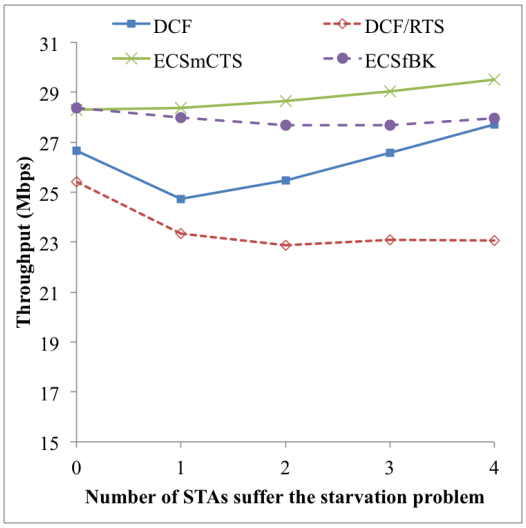


(a) Throughput

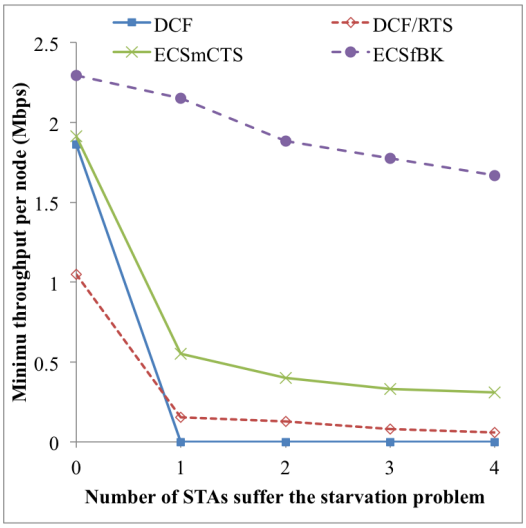


(b) Minimum throughput per node

Figure 60: Performance of ECSfBK in topologies with two-way hidden terminals

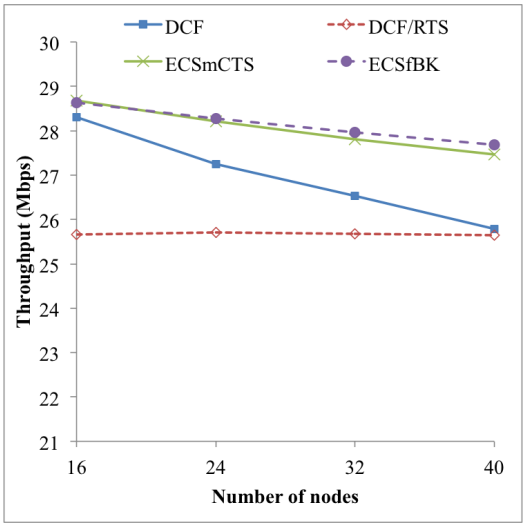


(a) Throughput

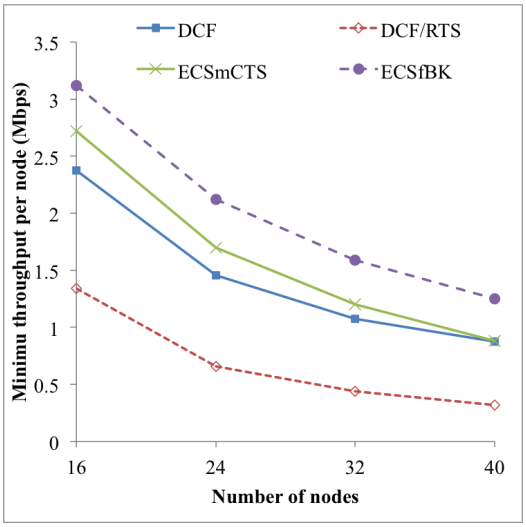


(b) Minimum throughput per node

Figure 61: Performance of ECSfBK in topologies with one-way hidden terminals



(a) Throughput



(b) Minimum throughput per node

Figure 62: Performance of ECSfBK in random topologies

CHAPTER VII

CONCLUSIONS

As one of the most popular last mile technologies, WiFi nowaday has been widely used in various applications. The goals of this research are to improve performance to support the growing requirements, and to enable controllability to support service differentiation and even performance prediction. Our work comprises three components considering the two goals:

First, we design mechanisms that give performance enhancement in the physical (PHY) layer with only upgrades in Access Points (APs). We present FastBeam, a set of algorithms that provide the benefits of beamforming to legacy nodes by adopting only the new APs. While existing techniques for RSSI-based beamforming severely under-perform when the channel changes rapidly, $FastBeam$ could dynamically adapt to the variation of the channel and still retain the advantages of RSSI-based beamforming. It requires only software modification on the client and is thus easy to deploy. Experimental evaluation shows that the throughput performance of $FastBeam$ is 1.4x on average and up to 1.8x in the best case, compared to the existing algorithm.

Second, consider future-proofing networks with a central controller, we propose algorithms that enable micro-level controllability to support service differentiation and even performance prediction. We presented Rhythm, a MAC protocol under a theme of *allowing WiFi network controllability without compromising its scalability*. We show that Rhythm enables the controllability of WiFi, subject to the constraints of (i) no fine-grained time synchronization and (ii) no additional hardware. Rhythm also has the properties of (i) low overhead, (ii) work conservation in the presence of non-backlogged nodes, and (iii) robustness to partial connectivity scenarios. We show that Rhythm provides near-optimal

channel utilization (200% improvement in high-density deployment) and weighted fairness under various traffic loads and connectivity scenarios through simulations and WARP experiments.

Finally, considering the backward compatibility issues of scheduled WiFi, we propose LWT. LWT realizes scheduled WiFi using purely distributed operations and provides good backward compatibility. Mechanisms to address practical problems of non-backlogged nodes, decodability vs. detectability, hidden terminals, backward compatibility, sleeping nodes and update of the schedule are proposed. With a WARP-based test-bed and ns-3, both experimental evaluations and simulation-based analysis are carried out to evaluate LWT. Evaluation results show that LWT achieves better channel efficiency, delay, adherence to schedule, and fairness comparing to related works.

To deal with hidden terminal problems, LWT uses a novel mechanism: *Switch*, *Switch* allows the transmitters and receivers of a transmission to exploit lightweight control channels *while the communication is ongoing*. We present the limitation and properties of *Switch*. We then use *Switch* as the core building block to solve three problems in WiFi networks: starvation due to hidden terminals, early collision termination, and frequency backoffs. We rely on WARP radios to experimentally verify that *Switch* is indeed possible and use *ns-3* simulations to study the impact of using *Switch* to solve the problems mentioned above. The simulation results show that *Switch* can solve the targeted problem with good efficiency. We believe that *Switch*, though simple, can have many potential benefits to be explored on many other applications.

CHAPTER VIII

FUTURE WORK

8.1 LWT in multiple collision domains

In Chapter 5, we proposed algorithms for scheduled WiFi, LWT, for WiFi networks in a single collision domain. Although WiFi networks typically have auto-channel-selection mechanisms (3 orthogonal channels in 2.4GHz and 25 orthogonal channels in 5GHz), and most networks can operate either in a single collision domain or are entirely disconnected, it is possible that solutions for multiple collision domains are required in future settings.

LWT can be extended to multiple collision domains by extending Transparent Transmissions to multiple collision domain. When multiple collision domains exist, LWT uses different flash signals to indicate the start and stop of transmission for each collision domain. Multiple transmissions can be scheduled in a schedule slot, and nodes increase Pos by one only after confirming the end of transmissions of all collision domains in the current schedule slot. This can be learned by either overhearing or receiving flash signals.

However, the challenge of this method lies in the identification of different collision domain. It is possible to identify the existence of different collision domain by overhearing and information exchange between APs. Collisions, especially repetitive collisions, can be an index of the presence of multiple collision domains. An AP can report such issue to the central controller and requests a new target schedule to deal with this situation. However, how to efficiently gather such information is still an open issue.

8.2 Scheduled WiFi and MIMO technologies

For achieving scheduled WiFi, we have developed algorithms: LWT and Rhythm. Below, we discuss how scheduled WiFi algorithms can operate with MIMO technologies.

- MIMO: Between one transmitter and one receiver, MIMO technology can be used to increase link reliability or spectral efficiency. For spectral efficiency, the transmitter can transmit different data streams simultaneously. On the other hand, the same data stream can be transmitted through different antennas to decrease error rate using spatial diversity. In both situation, since the MAC addresses of the transmitter and the receiver of each data packet do not change, the same portion of MAC header is delivered through different antennas to the receiver in both situations. Thus, the delivery of MAC addresses of the transmitter and receiver is more robust compared to traditional SISO. This property of MIMO helps nodes in the network to identify the current transmitting nodes without the help from LWT-CV. Thus, with MIMO transmissions (available in 802.11n/ac), it is possible that the implementation of LWT-CV can be omitted.

- MU-MIMO and interference mitigation: MU-MIMO is a MIMO technology in which a node with multiple antennas communicates with multiple nodes (each of them may have only a single antenna) simultaneously. Downlink MU-MIMO technology has been implemented in 802.11ac devices. On the other hand, uplink MU-MIMO is still an open issue. The Distributed Coordination Function (DCF) mode of WiFi is a random access control. Since each node determines the time of starting transmission randomly and independently, it is challenging to make multiple links transmit simultaneously to perform uplink MU-MIMO. The same problem applies to interference mitigation between multiple links. When using interference cancellation algorithms, the concerned links need to start transmitting at the same time, which is hard to achieve under DCF.

  It is possible for scheduled WiFi to fulfill this requirement. The central controller can schedule multiple links into one transmission slot with a predefined order specified in the target schedule. At the beginning of the transmission slot, nodes can start channel estimation process according to the specified order and exchange channel state

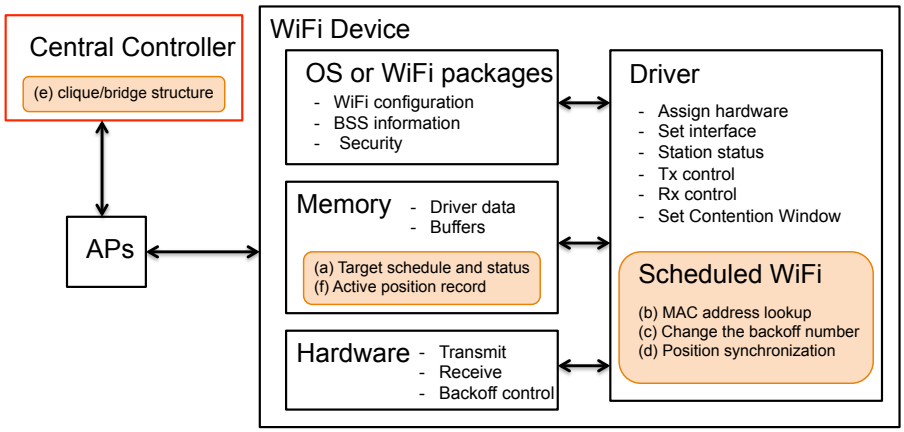


Figure 63: Software architecture of WiFi with implementation of scheduled WiFi on off-the-shelf WiFi devices

information (CSI) with each other for performing interference cancellation. Since the order of channel estimation is indicated in the target schedule, collisions won't happen even when multiple links are scheduled in the same slot. Other nodes that are not involved in the current slot stop the backoff mechanism once they overhear the channel estimation packets. Thus, they won't disturb the interference process during the channel estimation and information exchange. The scheduled links then can carry out interference cancellation algorithms or uplink MU-MIMO after the exchange of CSI information. We leave in-depth of exploration of transmission synchronization for future work.

8.3 Implementation of schedule WiFi in off-the-shelf devices

It is possible to build a viable prototype of schedule WiFi using off-the-shelf WiFi devices. Although different WiFi devices may have different software architectures, all of them support the same basic functionality. Figure 63 shows a general WiFi software architecture and the related basic functionality [9, 13, 30, 22, 53]. The required changes for implementing a prototype of scheduled WiFi are also shown in the figure. Scheduled WiFi can be implemented onto off-the-shelf WiFi devices as long as the required changes can be achieved. Below, we describe these basic implementation portions and leave detailed implementation

issues for future work. We consider two algorithms, LWT-WC, which is backward compatible, and Rhythm-Clique, which does not require possible hardware changes, and three scenarios that vary in their degree of modifiability and present the required changes for each of them: (i) modifications only on some APs in the network, (ii) modifications on all APs in the network, and (iii) modifications on all APs and clients.

(i) Can only change some APs: When we can only change some APs in the network, deploying LWT-WC algorithm among these APs can help with the scheduling of some downlink traffic. The central scheduler can insert DCF slots into the target schedule for legacy nodes. A simple RTS/CTS mechanism will be used to solve hidden terminal problems. The capabilities required for scheduled WiFi implementation in this scenario are the following (also indicated in Figure 63): (a) store the target schedule and current synchronization state in the memory, (b) look up MAC addresses of the current receiving packet for the first position match, (c) change backoff number according to the LWT basic algorithm, and (d) operate the position synchronization algorithm.

(ii) Can change all APs: Under this situation, in addition to deploying LWT-WC to all APs, we can implement Rhythm-Clique to deal with hidden terminals. APs can use NAV to control the activity of legacy nodes in the same way as starting a contention free period of PCF. The additional implementation requirements are: (e) support the clique and bridge structure in scheduled Rhythm-Clique when generating the target schedule, and (f) record extra status to determine if the current position is an active position.

(iii) Can change all APs and clients: This is an ideal scenario where both LWT-WC and Rhythm-Clique can be fully implemented.

8.4 Other Applications of Scheduled WiFi

In this thesis, we have explored the main application of scheduled WiFi including efficient high-density WiFi deployments, power-saving using precise sleep patterns, QoS using weighted fairness, and TCP-aware channel allocation. There are still other possibilities for adopting scheduled WiFi. We briefly describe each potential and leave in-depth evaluations for future work.

8.4.1 Access control for enterprises

The ability to have a fine-grained control of WLAN is desirable to enterprises when they set up networks for different usage. Enterprises usually have WiFi networks for primary business usage and some WiFi networks for visitors. Scheduled WiFi can help companies to have a full control of wireless resources deployment to the granularity of each transmission.

8.4.2 Internet of Things (IoT)

IoT has become a hot topic in recent years. An extension to the 802.11ah standard called WiFi $HaLow$ [16] is specifically targeted toward IoT environments. In many applications of IoT, the number of devices can be large. Although the throughput requirement of IoT is typically small, the channel efficiency decreases according to the number of contending nodes and is less related to the actual traffic amount. Scheduled WiFi is especially suitable to increase the channel efficiency by decreasing the collision rate among nodes.

8.4.3 Coexistence of WiFi and 5G networks

License-assisted access using LTE (LAA-LTE) has been proposed to operate LTE in unlicensed bands. One of the critical issues of LAA-LTE is how to coexist with WiFi efficiently. The primary trend of solving this problem is to use on-off duty cycles to separate LTE and WiFi transmissions in the time domain. During the on-cycle of LTE, WiFi devices are passively put to silence by frames containing large NAV transmitted from APs. The usage of NAV introduces overhead. Also, since broadcasting NAV stops all nodes in the nearby

area to transmit, nodes located nearby will stop transmit even if their transmission do not interfere current LTE transmissions. Thus, this method limits the flexibility of channel access control. On the other hand, by implementing scheduled WiFi, nodes can actively keep silence during the on-cycle of LTE. A more dedicated channel access control between LTE and WiFi also can be deployed by separating nodes in the same area into different Cliques.

REFERENCES

[1] "Cisco Wirless LAN Controller." http://www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/index.html?referring_site=smartnavRD.

[2] "Comcast, https://www.comcast.com."

[3] "Comcast shows off 1 gbps broadband, https://gigaom.com/2011/06/16/comcast-shows-off-1-gbps-broadband/."

[4] "Datasheet of WiFi transceiver MAX2829." http://datasheets.maximintegrated.com/en/ds/MAX2828-MAX2829.pdf.

[5] "Docsis 3.1, https://www.scribd.com/document/208506525/technology-d3-1140-docsis-session-dh."

[6] "Gartner, http://www.gartner.com/newsroom/id/2939217."

[7] "Internet users in the world, www.internetworldstats.com/stats.htm."

[8] "iPass WiFi Growth Map." http://www.ipass.com/wifi-growth-map/.

[9] "Linux Wireless." https://wireless.wiki.kernel.org/en/users/documentation.

[10] "ns-3." https://www.nsnam.org/.

[11] "ns-3: WiFi: The MAC model." https://www.nsnam.org/docs/models/html/wifi.html.

[12] "OpenFlow: Enabling Innovation in Your Network." http://www.openflow.org/.

[13] "OpenWrt." https://openwrt.org/.

[14] "Phocus array: Fidelity-comtech Inc." http://www.fidelity-comtech.com.

[15] "WARP." http://warpproject.org.

[16] "WiFi Halow." http://www.wi-fi.org/discover-wi-fi/wi-fi-halow.

[17] "Part 11:Amendment 1: Radio Resource Measurement of Wireless LANs," *IEEE Std. 802.11k-2008*, 2008.

[18] "Configuration and provisioning for wireless access points (CAPWAP) protocol specification," *IETF, RFC 5415*, 2009.

[19] "Light weight access point protocol," *IETF, RFC 5412*, 2010.

[20] "Part 11:Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std. 802.11-2012*, 2012.

[21] "Part 11:Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz," *IEEE Std. 802.11ac-2013*, 2013.

[22] ASRIADI, PRATOMO, I., AFFANDI, A., and RAHARDJO, D. S., "OpenVoice: Low-cost mobile wireless communication project for rural area based on OpenWRT," *International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 2015.

[23] BANSAL, T., CHEN, B., SINHA, P., and SRINIVASAN, K., "Symphony: Cooperative packet recovery over the wired backbone in enterprise WLANs," *ACM MobiCom*, 2013.

[24] BEJERANO, Y. and BHATIA, R. S., "MiFi: A Framework for Fairness and QoS Assurance for Current IEEE 802.11 Networks with Multiple Access Points," *IEEE/ACM TON*, 2006.

[25] CARCELO, J., BELLALTA, B., CANO, C., and OLIVER, M., "Learning-BEB: Avoiding Collisions in WLAN," *EUNICE*, 2008.

[26] CHENG, Y.-C., BELLARDO, J., BENKO, P., SNOEREN, A. C., VOELKER, G. M., and SAVAGE, S., "Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis," *ACM SIGCOMM*, 2006.

[27] CIDON, A., NAGARAJ, K., KATTI, S., and VISWANATH, P., "Flashback: Decoupled Lightweight Wireless Control," *ACM SIGCOMM*, 2012.

[28] CISCO. "https://meraki.cisco.com/solutions/high-density-wifi".

[29] DJUKIE, P. and MOHAPATRA, P., "Soft-TDMAC: A software TDMA-based MAC over commodity 802.11 hardware," *IEEE INFOCOM*, 2009.

[30] DUTT, S., HABIBI, D., and AHMAD, I., "A low cost Atheros System-on-Chip and OpenWrt based testbed for 802.11 WLAN research," *IEEE Region 10 Conference TENCON*, 2012.

[31] E., R., V., N., R., R., and S., R., "NAPman: Network-Assisted Power Management for WiFi Devices," *ACM MobiSys*, 2010.

[32] FATTAH, H. and LEUNG, C., "An Overview of Scheduling Algorithms in Wireless Multimedia Networks," *IEEE Wireless Communications*, 2002.

[33] FENG, W., NIU, Y., LI, Y., GAO, B., SU, L., JIN, D., and WU, D. O., "Throughput enhancement of IEEE 802.11ad through space-time division multiple access scheduling of multiple co-channel networks," *IEEE IET Communications*, 2015.

[34] GHADERI, M., SRIDHARAN, A., ZANG, H., TOWSLEY, D., and CRUZ, R., "TCP-Aware Channel Allocation in CDMA Networks," *TMC*, 2009.

[35] GOLD, R., "Optimal binary sequences for spread spectrum multiplexing," *IEEE Transactions on Information Theory*, 1967.

[36] GOLLAKOTA, S. and KATABI, D., "ZigZag decoding: Combating hidden terminals in wireless networks," *ACM SIGCOMM*, 2008.

[37] HAYKIN, S., *Communication Systems*. WILEY, 2001.

[38] HEMANTH, C. and VENKATESH, T. G., "Performance Analysis of Service Periods (SP) of the IEEE 802.11ad Hybrid MAC Protocol," *IEEE Transactions on Mobile Computing*, 2014.

[39] HEUSSE, M., ROUSSEAU, F., GUILLIER, R., and DUDA, A., "Idle sense: An optimal access method for high throughput and fairness in rate diverse wireless LANs," *ACM SIGCOMM*, 2005.

[40] J., L. and L., Z., "Micro Power Management of Active 802.11 Interface," *ACM MobiSys*, 2008.

[41] J., M. and R., C. R., "Avoiding the Rush Hours: wiFi Energy Management via Traffic Isolation," *ACM MobiSys*, 2011.

[42] JAMIESON, K. and BALAKRISHNAN, H., "PPR: Partial Packet Recovery for Wireless Networks," *ACM SIGCOMM*, 2007.

[43] JI, X., WANG, J., LIU, M., YAN, Y., YANG, P., and LIU, Y., "Hitchhike: Riding Control on Preambles," *IEEE INFOCOM*, 2014.

[44] KIM, J., KWON, S.-C., and CHOI, G., "Performance of Video Streaming in Infrastructure-to-Vehicle Telematics Platforms with 60 GHz Radiation and IEEE 802.11ad Baseband," *IEEE Transactions on Vehicular Technology*, 2016.

[45] LAKSHMANAN, S., SUNDARESAN, K., RANGARAJAN, S., and SIVAKUMAR, R., "Practical Beamforming based on RSSI Measurements using Off-the-shelf Wireless Clients," *ACM IMC*, 2009.

[46] LAKSHMANAN, S., SUNDARESAN, K., KOKKU, R., KHOJESTEPOUR, A., and RANGARAJAN, S., "Towards Adaptive Beamforming in Indoor Wireless Networks: An Experimental Approach," *IEEE INFOCOM (Miniconference)*, 2009.

[47] LAKSHMANAN, S., SUNDARESAN, K., RANGARAJAN, S., and SIVAKUMAR, R., "The Myth of Spatial Reuse with Directional Antennas in Indoor Wireless Networks," *PAM*, 2010.

[48] LIU, X., SESHAN, S., and STEENKISTE, P., "When Are Directional Antennas Useful in Indoor Environments?," *WiNECH*, 2011.

[49] LIU, X., SHETH, A., KAMINSKY, M., PAPAGIANNAKI, K., SESHAN, S., and STEENKISTE, P., "DIRC: Increasing Indoor Wireless Capacity Using Directional Antennas," *ACM SIGCOMM*, 2009.

[50] LIU, X., SHETH, A., KAMINSKY, M., PAPAGIANNAKI, K., SESHAN, S., and STEENKISTE, P., "Pushing the Envelope of Indoor Wireless Spatial Reuse using Directional Access Points and Clients," *ACM MobiCom*, 2010.

[51] MAGISTRETTI, E., GUREWITZ, O., and KNIGHTLY, E. W., "802.11ec: Collision Avoidance without Control Messages," *ACM MobiCom*, 2012.

[52] NAVDA, V., SUBRAMANIAN, A. P., DHANASEKARAN, K., TIMM-GIEL, A., and DAS, S. R., "MobiSteer: Using Steerable Beam Directional Antenna for Vehicular Network Access," *ACM MobiSys*, 2007.

[53] PALAZZI, C. E., BRUNATI, M., and ROCCETTI, M., "An OpenWRT solution for future wireless homes," *IEEE International Conference on Multimedia and Expo (ICME)*, 2010.

[54] PARK, E.-C. and KIM, H., "TCP-aware bidirectional bandwidth allocation iin IEEE 802.16 networks," *Wireless Network*, 2010.

[55] PAULRAJ, A., NADAR, R., and GORE, D., "Introduction to Space-Time Wireless communications," *Cambridge University Press*, May 2003.

[56] QIAO, D. and SHIN, K. G., "Achieving Efficient Channel Utilization and Weighted Fairness for Data Communications in IEEE 802.11 WLAN under the DCF," *Quality of Service*, 2002.

[57] RAMACHANDRAN, K., KOKKU, R., SUNDARESAN, K., GRUTESER, M., and RANGARAJAN, S., "R2D2: Regulating Beam Shape and Rate as Directionality meets Diversity," *ACM MobiSys*, 2009.

[58] RUCKUS. "http://c541678.r78.cf2.rackcdn.com/appnotes/bpg-highdensity.pdf".

[59] SANABRIA-RUSSO, L., GRINGOLI, F., BARCELO, J., and BELLALTA, B., "Implementation and Experimental Evaluation of a Collision-Free MAC Protocol for WLANs," *IEEE ICC*, 2015.

[60] SEN, S., CHOUDHURY, R. R., and NELAKUDITI, S., "No Time to Countdown: Migrating Backoff to the Frequency Domain," *ACM MobiCom*, 2011.

[61] SEN, S., CHOUDHURY, R. R., and NELAKUDITI, S., "CSMA/CN: Carrier Sense Multiple Access With Collision Notification," *IEEE Transction on Networking*, April 2012.

[62] SHIH, C.-F., JIAN, Y., and SIVAKUMAR, R., "Look Who's Talking: A Practical Approach for Achieving Scheduled WiFi in a Single Collision Domain," *ACM International Conference on emerging Networking EXperiments and Technologies*, 2015.

[63] SHIH, C.-F., KRISHNASWAMY, B., JIAN, Y., and SIVAKUMAR, R., "Scheduled WiFi Using Distributed Contention in WLANs: Algorithms, Experiments, and Case-Studies," *ACM/Springer Wireless Networks Journal (WINET)*, 2016.

[64] SHIH, C.-F., KRISHNASWAMY, B., and SIVAKUMAR, R., "Rhythm: Achieving Scheduled WiFi Using Purely Distributed Contention in WLANs," *IEEE GlobeCom*, 2015.

[65] SHIH, C.-F. and SIVAKUMAR, R., "FastBeam: Practical Fast Beamforming for Indoor Environments," *IEEE ICNC*, 2014.

[66] SHRIVASTAVA, V. and OTHERS, "CENTAUR: Realizing the full potential of centralized WLANs through a hybrid data path," *ACM MobiCom*, 2009.

[67] SHRIVASTAVA, V., RAYANCHU, S., BANERJEE, S., and PAPAGIANNAKI, K., "PIE in the sky: Online passive interference estimation for enterprise WLANs," *NSDI*, 2011.

[68] TAN, K., FANG, J., ZHANG, Y., CHEN, S., SHI, L., ZHANG, J., and ZHANG, Y., "Fine Grained Channel Access in Wireless LAN," *ACM SIGCOMM*, 2010.

[69] VESTIN, J., DELY, P., KASSLER, A., BAYER, N., EINSIEDLER, H., and PEYLO, C., "CloudMAC: towards Software Defined WLANs," *ACM SIGMOBILE*, 2013.

[70] VUTUKURU, M., BALAKRISHNAN, H., and JAMIESON, K., "Cross-layer wireless bit rate adaptation," *ACM SIGCOMM*, 2009.

[71] WU, K., TAN, H., LIU, Y., ZHANG, J., ZHANG, Q., and NI, L. M., "Side Channel: Bits over Interference," *IEEE Transction on Mobile Computing*, August 2012.

[72] XIAO, Y., SAVOLAINEN, P., and KARPPANEN, A., "Practical power modeling of data transmission over 802.11g for wireless applications," *IMC*, 2010.

[73] YANG, X. and VAIDYA, N., "A wireless MAC protocol using implicit pipelining," *IEEE Transactions on Mobile Computing*, 2006.

[74] YU, H., ZHONG, L., SABHARWAL, A., and KAO, D., "Beamforming on Mobile Devices: A First Study," *ACM MobiCom*, 2011.

[75] ZENG, Z., GAO, Y., TAN, K., and KUMAR, P. R., "Chain: Introducing minimum controlled coordination into random access MAC," *IEEE INFOCOM*, 2011.

[76] ZHANG, J., SHEN, H., TAN, K., CHANDRA, R., ZHANG, Y., and ZHANG, Q., "Frame Retransmissions Considered Harmful: Improving Spectrum Efficiency Using Micro-ACKs," *ACM MobiCom*, 2012.

[77] ZHANG, X. and SHIN, K. G., "E-MiLi: Energy-Minimizing Idle Listening in Wireless Networks," *IEEE TRANSACTIONS ON MOBILE COMPUTING*, 2012.

[78] ZHOU, W., LI, D., SRINIVASAN, K., and SINHA, P., "Domino: Relative scheduling in enterprise wireless LANs," *ACM CoNEXT*, 2013.